# 可视化体系结构描述语言XYZ/ADL*

骆华俊　唐稚松　郑建丹

(中国科学院软件研究所计算机科学开放研究实验室　北京　100080)

E-mail:zheng@ox.ios.ac.cn

**摘要**　提出一种基于时序逻辑语言XYZ/E的可视化体系结构描述语言XYZ/ADL(XYZ/architecture description language).它采用组件、连接件及交互端等设计单元,能描述常用的多种软件体系结构.XYZ/E能以统一的形式同时表示静态语义和动态语义,因此,以XYZ/E为基础的XYZ/ADL能在统一框架下完成不同抽象级体系结构设计之间的逐步过渡.

**关键词**　时序逻辑语言,软件体系结构,体系结构描述语言,组件,连接件,交互端.

**中图法分类号**　TP311

随着软件系统规模和复杂性的不断增大,传统的软件设计模式已不能适应要求.为了解决这一问题,近年来,国际软件工程界学者(如CMU的D.Garlanm,M.Shaw等人)提出了研究软件体系结构SA(software architecture)这一新方法[1,2].该方法主要着眼于软件系统的全局组织形式,即软件体系结构,在高层次上把握系统各部分之间的内在联系,并从全局的、整体的角度去理解和分析整个系统的行为和特性,有助于解决当前开发复杂的大型软件所存在的困难.其中,软件体系结构描述语言ADL(architecture description language)是SA研究的一个核心问题,它不但是形式化描述SA的基本工具,而且也是对SA进行求精、验证、演化和分析的前提和基础.

XYZ/E[3,4]作为基于线性时序逻辑系统的一种面向软件工程的时序逻辑语言,可在统一的框架下,既能表示适应冯·诺依曼(Von Newmann)体系的状态转换机制的命令式语言,又能表示适应逻辑推理特征的直言式公式语言,非常适合于描述软件体系结构[5].基于此,我们针对XYZ/E的特点,提出了可视化软件体系结构描述语言XYZ/ADL,以支持软件系统的体系结构设计.

本文首先介绍XYZ/ADL中的基本设计单元,然后说明如何组织这些设计单元来表示不同的体系结构,最后是与相关工作的比较以及对全文的总结.

## 1　基本设计单元

XYZ/ADL由以下主要设计单元组成:组件(component)、连接件(connector)和交互端(port).用户通过对这些设计单元的操作组合来设计软件的体系结构.

### 1.1　组　件

简单地讲,组件就是具有一定功能的逻辑单元或者是用户级的逻辑对象,比如外部文件或共享数据等等,是XYZ/ADL中最基本的设计单元.每一个组件由规范和内部结构实现两部分组成.其中,规范用于描述组件的逻辑功能,它刻画了组件"做什么",我们一般用Pre-Post断言[3]来表示.在软件开发的高层设计中,组件的规范有可能非常抽象,为了使它能够平滑地过渡到底层的代码实现,用某种体系结构(比如管道-过滤器结构等)来表示

* 本文研究得到国家"九五"重点科技攻关项目基金(No.98-780-01-07-01)和国家863高科技项目基金(No.863-306-ZT02-04-01)资助.作者骆华俊,1976年生,硕士,主要研究领域为软件工程工具.唐稚松,1925年生,研究员,博士生导师,中国科学院院士,主要研究领域为计算机科学理论,软件工程.郑建丹,女,1977年生,硕士生,主要研究领域为软件工程工具.

本文通讯联系人:郑建丹,北京100080,中国科学院软件研究所计算机科学开放研究实验室

本文2000-01-17收到原稿,2000-04-13收到修改稿

© 中国科学院软件研究所　http://www.jos.org.cn

组件的内部实现.它包括下层的组件以及它们之间的交互,这里,主要用图形方式来表示.结合XYZ/E的文本描述,刻画了"怎么做".由规范向内部结构实现的转换,即构成软件开发过程中的一步过渡(transition),这样的逐层过渡就构成了开发软件系统的逐步求精过程.对于XYZ/ADL而言,每当描述完一个组件的内部结构实现之后,可自动或半自动地生成描述体系结构语义的XYZ/E程序.并根据这个语义程序以及组件的规范进行语义一致性检验,从而将XYZ系统中原有的两种支持软件开发的方法,即支持模块化程序设计的方法和支持从规范到算法实现的逐步求精方法有机地结合起来[3].

在XYZ/ADL中,我们对组件进行适当的分类,使它与XYZ/E程序中的构件相对应,以便自动生成程序.当前,组件包括赋值(assignment)、条件(condition)、规范(specification)和共享数据(shared data)等等,在可视化环境中,分别对应于矩形、椭圆形、菱形和双矩形等图形元素.

### 1.2 连接件

连接件是软件体系结构中的一个非常重要的概念,它表示组件之间的交互方式.也就是说,连接件定义了组件之间交互的规则并且给出了一些实现的机制.连接件一般并不对应于一段XYZ/E代码,它可以有非常灵活的表示,比如,控制流、数据流、信息流、过程调用、事件、表的入口、缓冲区、动态数据结构、管道或者更一般的协议.

连接件包括外部界面、内部实现和交互单元3个部分.它的外部界面被称作协议(protocol),刻画了这个连接件所表示的交互的规则.与组件一样,如果描述交互的协议太抽象,我们就用某种结构来表示它的内部实现.连接件所表示的交互行为总是发生在它的交互单元之间,比如,"过程调用"总是从"调用者"向"被调用者"发起调用.当连接件连接组件的时候,它的交互单元实际上就是后面要讲的交互端.因此,一般并不显式地说明连接件的交互单元.

在实际的实现中,我们也对连接件进行分类,用各种不同的带箭头的线段来表示.在当前的XYZ/ADL版本中,只定义了一些简单的和比较常用的连接件,包括控制流(control flow)、信息流(information flow)、过程调用(procedure call)、隐式调用(implicit invocation)、管道(pipe)、客户-服务器协议(client-server protocol)以及分层协议(layer protocol)等等.

### 1.3 交互端

连接件通过交互端连接不同的组件,而组件通过各自的交互端与环境进行交互.实际上,交互端就是组件对外交互部分的一个抽象,我们在可视化环境中,用依附在表示组件的矩形(或其他图形元素)上的小圆圈或小椭圆来表示.考虑一个简单的客户机-服务器的例子,客户程序向服务器请求服务,并等待服务器发送回结果,从而完成所需执行的功能.我们注意到客户程序请求服务并等待结果这个行为就是和环境(这里指的是服务器)发生了交互,我们把这种行为抽象出来,并定义为"客户"交互端.相应地,服务程序也有一个"服务器"交互端.在软件系统的高层设计中,作这样的抽象可以避免一些底层实现的细节,并且又能很清楚地刻画出组件与外界交互的过程,便于对系统的行为进行分析、设计以及验证等等.

交互端总是依附在某个组件上,它描述了这个组件对外交互的方式和行为.在上面的客户机-服务器的例子中,如果把"客户"交互端依附在组件上,组件的行为就是前面所描述的那样:向服务器请求数据,等待服务器返回结果,完成计算.但是,如果组件包含的是"输出"交互端而不是"客户"交互端,那么组件的行为就是:完成计算,把结果输出给其他组件.这时,由这种组件所构成的体系结构就变成下面第2节将要介绍的过滤器-管道体系结构,因此,添加不同的交互端很可能导致完全不同的设计.

一般来说,交互端都是对应的,如"客户"交互端总是对应于"服务器"交互端,而"输入"交互端则总是对应于"输出"交互端.实际上,这些交互端之间交互的行为规则正是在第1.2节所介绍的连接件.交互端是一个逻辑概念,可以把它理解成组件对外交互的作用点,它并不一定能够很清晰地在最终的程序代码中表现出来,而像一些较复杂的交互端很可能隐藏在组件的代码实现中.

这样,以一定的方式组合不同的图形元素,构成具有某种拓扑结构的一个有向图.在这个图中,顶点代表组件,边代表一个连接件,整个有向图反映了一个程序或系统的构件的组织结构、它们之间的关联关系(连接件)以及支配系统设计和演变的原则和方针.我们用这种有向图来表示上层组件的内部结构实现.它由一组组件、连接件以及组件和连接件如何结合在一起的约束限制的描述组成.这种图形表示能加深用户对体系结构的直观理

© 中国科学院软件研究所 http://www.jos.org.cn

解;同时,系统可自动或半自动生成表示该体系结构语义的 XYZ/E 程序,用户可以根据这个语义程序在 XYZ 系统中对该结构进行分析和验证.

## 2　体系结构表示

在软件设计上,软件工程师经过多年的理论探索和工程实践,逐渐发展了一系列构造复杂软件系统的方法、技术、模式和风格,CMU 的 D. Garlan,M. Shaw 等专家把一组体系结构实例所具有的共同的结构模式抽象出来,称为一种软件体系结构风格[1]. 例如,"管道-过滤器"体系结构风格是一种具有增量式的、流转换的系统体系结构. 一种体系结构风格决定了组件、连接件和一组如何将它们结合在一起的约束限制,即决定了一种体系结构的框架. 文献[1]对各种常用的体系结构风格进行了系统的分类:常用的体系结构风格主要有管道-过滤器体系结构(pipe-filter architecture)、分层系统(layered system)、面向对象结构(object-oriented organization)、基于事件/隐式调用的系统(event-based/implicit invocation system)和分布式进程等;其他特殊体系结构,包括特定领域的体系结构(domain-specific architecture)(如文献[6]中所介绍的军事领域的软件体系结构)和异质的体系结构(heterogeneous architecture)即包含不同类型的体系结构.

由于 XYZ/E 同时又是一种可执行的编程语言,可直接用于最后的算法实现,因此,我们加入另外一种最底层的体系结构风格:流程图结构. 在 XYZ/ADL 中,我们能以图形的方式表示上述常用的体系结构风格. 下面,我们将说明其中的 3 种.

### 2.1　流程图结构

流程图结构(flowchart form structure)是最基本的一类体系结构,它直接表示了冯·诺依曼体系结构的程序状态转换机制. 流程图结构包括了赋值表达式、条件表达式和规范表达式 3 种组件,组件之间用控制流连接,表示执行的顺序关系. 每一个控制流连接件都带有一个路标,它对应于 XYZ/E 程序中每个条件原子式的标号(在下面的流程图例子(如图 1 所示)中,每个组件左边的路标对应于条件原子式的定义标号[8],而右边的标号则对应于转出标号. 同一个标号可以连接到不同的条件表达式组件中). 整个流程图的结构要求是确定的,即每个组件完成计算后下一步的走向是唯一确定的. 同时,控制流所带的标号以及各条件表达式都应满足 XYZ/E 程序的规定. 流程图结构直接对应于确定的串行 XYZ/E 程序,适合于软件系统中的下层模块,在 XYZ/ADL 中能够方便地自动生成它所对应的 XYZ/E 程序,用户可以在 XYZ 系统中检查这个程序是否满足该流程图结构所代表的组件的规范.

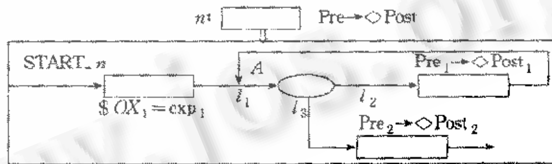


Fig. 1　Flowchart form structure
图1　流程图结构

图 1 是流程图结构的一个例子,根据该图自动生成的 XYZ/E 程序如下:

对应的基本 XYZ/E 程序(XYZ/BE[3]):

$[LB=\mathrm{START_}n \Rightarrow \$Ox_1=\mathrm{exp}_1 \wedge \$OLB=l_1;$

$LB=l_1 \wedge A \Rightarrow \$OLB=l_2;$

$LB=l_1 \wedge \sim A \Rightarrow \$OLB=l_3;$

$LB=l_2 \wedge \mathrm{Pre}_1 \Rightarrow \Diamond(\mathrm{Post}_1 \wedge LB=l_1);$

$LB=l_3 \wedge \mathrm{Pre}_2 \Rightarrow \Diamond(\mathrm{Post}_2 \wedge LB=\mathrm{STOP})]$

结构化高级语言语句形式 XYZ/SE[3]程序:

$[LB=\mathrm{START_}n \Rightarrow \$Ox_1=\mathrm{exp}_1 \wedge \$OLB=l_1;$

$*[LB=l_1 \wedge A \Rightarrow \$OLB=l_2 \mid \$OLB=\mathrm{EXIT};$

© 中国科学院软件研究所　http://www.jos.org.cn

$LB=l_2 \wedge \mathrm{Pre}_1 \Rightarrow \Diamond(\mathrm{Post}_1 \wedge LB=l_1)]$

$LB=l_3 \wedge \mathrm{Pre}_2 \Rightarrow \Diamond(\mathrm{Post}_2 \wedge LB=\mathrm{STOP})]$

### 2.2 管道-过滤器体系结构

管道-过滤器体系结构(pipe-filter architecture)由过滤器组件和管道连接件组成，每个组件包括一个“输入”交互端和一个“输出”交互端，组件通过“输入”交互端读进所需的数据进行加工处理，再把处理的结果通过“输出”交互端输出.管道连接件依次把一个组件的“输出”端连接到下一个组件的“输入”端，表示把前一个组件计算的结果输送给后一个组件.这样，数据从第1个组件读入加工后沿着管道连接件依次流入下一个组件进行加工处理，直到最后一个组件输出结果.

过滤器组件的设计是完全独立的，并不受前面或后面的组件的影响，因此不必知道它前面或后面的组件是什么.一个过滤器组件的设计只是根据它的规范保证什么样的输入将会导致什么样的输出，这样做的好处是可以对组件进行重用.管道连接件用来传输具有某种类型的数据，并保证一端的输入能正确地在另一端输出，这里的正确包括数据的内容和次序都保持一致.整个结构是线性的，即不允许在结构图中出现环.

管道-过滤器体系结构着重于描述数据流的传输，系统整体的行为可以看作是所有过滤器的并行执行，因此，可用 XYZ/E 中的并行语句来实现.

图2是管道-过滤器体系结构的一个例子，它所对应的 XYZ/E 程序为

$$LB=l \Rightarrow \| [f_1, f_2, \ldots, f_k].$$

其中 $f_i$ 的前置条件和后续断言分别用 $\mathrm{Pre}_i$ 和 $\mathrm{Post}_i$ 来表示.

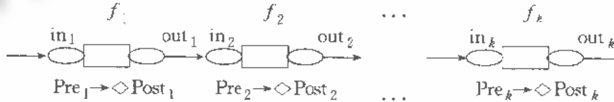


Fig. 2 Pipe-Filter architecture
图2 管道-过滤器体系结构

### 2.3 客户机-服务器体系结构

客户机-服务器体系结构(client-server architecture)是分布式程序设计的一种常用的组织结构，在这种结构中，有一个特殊的进程即服务器向其他客户进程提供服务，每一个客户进程都知道服务器的地址，可以通过远程过程调用与服务器通信；但服务器不必知道客户进程的身份.图3给出的，就是一个客户机-服务器体系结构.

在 XYZ/ADL 中，客户机-服务器体系结构包括了客户和服务器组件，图3中的 Sever，Client 1 和 Client 2.从本质上讲，它们的差别只是在于功能不同，因此，在 XYZ/ADL 中并不区分这两种组件，而把区分留给用户自己，在这里这么称呼只是为了后面叙述的方便.客户组件包含一个“客户”交互端，服务器组件包括一个“服务器”交互端，“客户-服务器协议”描述了上述通信过程，它总是连接了“客户”端和“服务器”端，表示客户程序与服务器之间的信息交互.

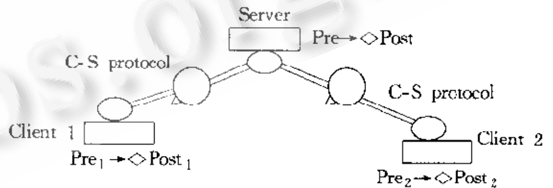


Fig. 3 Client-Server architecture
图3 客户机-服务器体系结构

在 XYZ/ADL 中，假定对于分布式环境下运行的相互通信的用户程序，存在一个元程序，其中包含以这些相互通信的用户程序为其组成单元的元并行语句，每一个用户程序事实上只在这个元程序的元并行语句中扮演一个“通信进程”的角色.因此，从实现的角度看，分布式环境下的用户程序与一般并发环境下的通信进程在形式上是一样的.一般来讲，客户机-服务器体系结构都是作为整个系统的第1层结构，它的交互协议有可能比较复杂，而且很多情况下在细节上有所差别，比如，有的结构要求对服务器进行初始化而有的则不要求，因此我们要求用户自己利用 XYZ/E 对协议进行描述.其实，这种高层的体系结构设计，其主要意义并不在于如何自动地生成程序(相反地，我们更多地要求用户提供一些程序的实现)，而主要是使用户对整个系统的结构有更好的理解以及对自己的设计思想的一个整理，以便于作进一步的设计.当然，当用户在 XYZ/ADL 中设计好一个客户机-服务

© 中国科学院软件研究所 http://www.jos.org.cn

器的例子后,可以把它保存下来作为一个模板,在以后类似的设计中重用.

## 3 与相关工作的比较

现有的体系结构描述语言(如 Wright[7],Rapide[8]等)一般都使用比较抽象的规范语言(如 CSP[9],Z[10])来描述体系结构,但由于这些规范语言只是描述性的,因此,它们一般都把对组件的规范描述和对组件的结构实现截然分开,难以对上层规范进行逐步求精.而 XYZ/E 作为基于线性时序逻辑系统的一种面向软件工程的时序逻辑语言,既能表示程序的静态语义,又能表示动态语义[3],避免了 CSP 和 Z 语言中只能描述其中一个的不足[5],而且在 XYZ 系统中,还有一些检验语义一致性的方法和工具(如 XYZ/VERI),有助于逐步求精过程中的语义一致性检验.因此,以 XYZ/E 为其文本描述的 XYZ/ADL 不仅能够对软件体系结构进行形式化描述,而且能对其进行求精、验证、演化和分析,在统一的框架下描述组件的规范和结构实现,并逐步过渡到最后的算法实现,从而能够支持软件开发设计的全过程.

## 4 总 结

XYZ/ADL 作为一种可视化体系结构描述语言,结合 XYZ/E 文本描述,能够描述各种不同的软件体系结构,并自动或半自动地生成相应的 XYZ/E 程序.用它可以逐层地进行体系结构设计.第 1 层体系结构包括若干组件以及它们之间的交互,对这一层包含的组件作进一步的分解(如果有必要的话),形成第 2 层的设计,就这样一层一层地分解设计下去,直到最后的代码实现.因此,从整体上看,XYZ/ADL 可以描述一个复杂的体系结构,它包含了许多不同的体系结构类型.相对于其他体系结构描述语言,XYZ/ADL 可在统一框架下描述和分析体系结构的静态语义和动态语义.

在 XYZ 系统中,我们建立了一个可视化工具,用于支持 XYZ/ADL,由于体系结构的复杂性,目前这个工具还有许多需要完善的地方,比如对某些较复杂的连接协议,还不能提供验证工具,并且对体系结构的逐层分解缺乏语义分析工具等等,今后,我们将在这些方面作进一步的研究.

### 参考文献


1 Shaw M, Garlan D. Software Architecture: Perspectives on an Emerging Discipline. Englewood Cliffs, NJ: Prentice Hall, Inc., 1996

2 Garlan D, Perry D E. Introduction to the special issue on software architecture. IEEE Transactions on Software Engineering, 1995,21(4):269~274

3 Tang Zhi-song. Temporal Logic Programming and Software Engineering. Beijing: Science Press, 1999
(唐稚松.时序逻辑程序设计与软件工程.北京:科学出版社,1999)

4 Tang Zhi-song. A temporal logic language oriented toward software engineering——an introduction to XYZ system. Chinese Journal of Advanced Software Research, 1994,1(1):1~29

5 Zhou Ying-xin, Ai Bo. A study of software architecture modeling. Journal of Software, 1998,9(11):866~872
(周莹新,艾波.软件体系结构建模研究.软件学报,1998,9(11):866~872)

6 Allan T, Frederick H-R, Lee Erman *et al.* Overview of Teknowledge's domain-specific software architecture program. ACM SIGSOFT Software Engineering Notes, 1994,19(4):68~76

7 Allen R, Garlan D. Formal connectors. Technology Report, CMU-CS-94-115. Pittsburgh: Carnegie Mellon University, 1994

8 Luckham D, Lary C, Augustin M *et al.* Specification and analysis of system architecture using rapide. IEEE Transactions on Software Engineering, 1995,21(4):336~355

9 Hoare C A R. Communicating Sequential Process. Englewood Cliffs, NJ: Prentice-Hall, Inc., 1985

10 Sprivey J. The Z Notation: a Reference Manual. Englewood Cliffs, NJ: Prentice Hall, Inc., 1989


© 中国科学院软件研究所 http://www.jos.org.cn

# Visual Architecture Description Language XYZ/ADL

LUO Hua-jun TANG Zhi-song ZHENG Jian-dan

(*Laboratory of Computer Science Institute of Software The Chinese Academy of Sciences Beijing* 100080)

**Abstract** In this paper, the authors describe a visual language XYZ/ADL (XYZ/architecture description language) for software architecture description based on XYZ/E to support a new approach of architecture-based programming. XYZ/ADL is composed of components, connectors and ports, by which it can describe many commonly used kinds of software architectures. Since XYZ/E can represent both statistic and dynamic semantics in a unified form, XYZ/ADL which is built on it can finish smoothly the transformation between architectures at different abstract levels in a uniform frame.

**Key words** Temporal logic language, software architecture, architecture description language, component, connector, port.

© 中国科学院软件研究所 http://www.jos.org.cn