软件体系结构建模研究*

周莹新 艾 波

(北京邮电大学计算机科学与技术学院 北京 100088)

E mail: yxzhou@mii.gov.cn

摘要 提出了软件体系结构工程的概念,建立了软件体系结构生命周期模型,并对软件体系结构进行了分类和建模,介绍了几种典型的软件体系结构语言.提出了一个基于时序逻辑的形式化体系结构语言——XYZ/SAE.该语言可作为系列化时序逻辑语言族XYZ/E的子语言,支持对软件体系结构中构件、连接件和配置的描述,并可在统一的框架下描述软件体系结构的静态行为和动态行为.

关键词 软件体系结构,软件体系结构建模,软件体系结构建模语言,时序逻辑,构件,连接件.

中图法分类号 TP311

开发大型软件存在的困难始于60年代,从那时起,软件设计从侧重于编码阶段转向侧重于结构化设计.70年代软件工程的思想和方法得到了广泛的应用,由于设计是与实现相对独立的过程,需要专用的符号、技术和工具,兴起了开发CASE(computer aided support environment)工具热.80年代,软件工程领域的研究重心从特定的软件设计转移到集成设计和设计的过程,以至更广泛的软件过程和管理.随着软件系统愈来愈大,愈来愈复杂,软件设计的核心已经超越了传统的"算法+数据结构=程序"的计算设计模式,取而代之的是系统的总体结构的设计和规范.Dewayne E. Perry和Alexander L. Wolf[1]认为90年代是研究软件体系结构的时代.

那么,什么是软件体系结构呢?根据文献[2]的定义,软件体系结构是一个程序或系统的构件的组织结构、它们之间的关联关系以及支配系统设计和演变的原则和方针.一般地,一个系统的软件体系结构由一组计算构件、构件之间的交互-连接件以及构件和连接件如何结合在一起的约束限制的描述组成.如果用图形表示,则节点可以表示构件;边或弧表示连接件.约束限制可能是体系结构的拓扑限制(如:无循环)以及有关语义等方面的限制.

1 软件体系结构的生命周期模型

目前,软件体系结构的研究已发展为软件工程领域的一个独立的学科分支,需要有严格的理论基础和工程原则.[3]为此,本文提出软件体系结构工程的概念,并对软件体系结构的生命周期建模,从而使软件体系结构的各个研究方向和内容有机地统一在一起,使得在以过程为中心的兆程序设计中起关键作用的体系结构既具有严格的理论基础,又具有严格的工程原则.

定义1. 软件体系结构工程=形式化模型+软件技术+软件工程

定义2. 软件体系结构生命周期模型是对软件体系结构在整个生存期间所需经历的所有阶段和步骤的描述,这种描述独立于具体的体系结构,使得体系结构的设计遵循一定的理论基础和工程原则.

为了形象化地表示体系结构的生命周期,本文建立了一个软件体系结构生命周期模型,如图1所示,它主要由以下几个阶段组成:

(1) 软件体系结构的非形式化描述(Software Architecture Informal Description).一种软件体系结构在其产生时,其思想通常是简单的,并常常由软件设计师用非形式化的自然语言表示概念、原则.例如:客户机-服务器体系结构就是为适应分布式系统的要求,从主从式演变而来的一种软件体系结构.尽管该阶段的描述常是用自然语言描述的,但是该阶段的工作却是创造性和开拓性的.

(2) 软件体系结构的规范描述和分析(Software Architecture Specification and Analysis).这一阶段通过运用合适

---

* 作者周莹新,1970年生,博士,主要研究领域为软件体系结构,通信软件体系结构.艾波,1958年生,博士,教授,博导,主要研究领域为通信软件工程.

本文通讯联系人:周莹新,北京 100804,北京市西长安街13号邮电部信息中心

本文1997-06-09收到原稿,1997-10-05收到修改稿

的形式化数学理论模型对第 1 阶段的体系结构的非形式化描述进行规范定义,从而得到软件体系结构的形式化规范描述,以使软件体系结构的描述精确、无歧义,并进而分析软件体系结构的性质,如无死锁性、安全性、活性等.分析软件体系结构的性质有利于在系统设计时选择合适的软件体系结构,从而对软件体系结构的选择起指导作用,避免盲目选择.

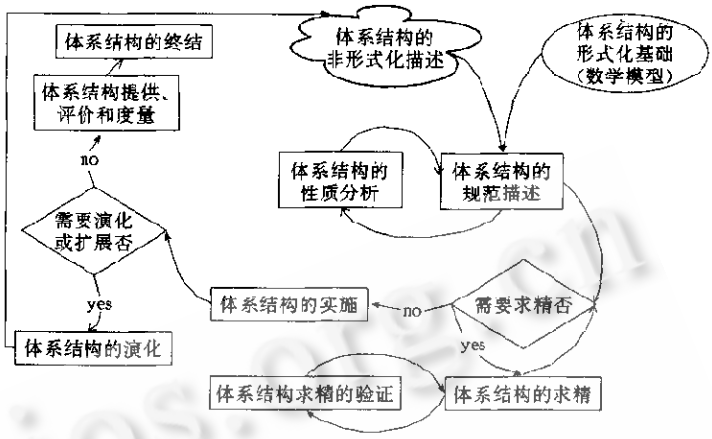

图1　软件体系结构的生命周期模型

(3) 软件体系结构的求精及其验证(Software Architecture Refinement and Its Verification).大型系统的软件体系结构总是通过从抽象到具体,逐步求精而达到的,因为一般说来,由于系统的复杂性,抽象是人们在处理复杂问题和对象时必不可少的思维方式,软件体系结构也不例外.但是过高的抽象却使软件体系结构难以真正在系统设计中实施.因而,如果软件体系结构的抽象粒度过大,就需要对体系结构进行求精、细化,直至能够在系统设计中实施为止.在软件体系结构的每一步求精过程中,需要对不同抽象层次的软件体系结构进行验证,以判断较具体的软件体系结构是否与较抽象的软件体系结构的语义一致,并能实现抽象的软件体系结构.我们这里并不排斥在不求精的情况下对软件体系结构的验证,而只是侧重于研究和讨论软件体系结构的求精及其验证.

(4) 软件体系结构的实施(Software Architecture Enactment).这一阶段将求精后的软件体系结构实施于系统的设计中,并将软件体系结构的构件和连接件等有机地组织在一起,形成系统设计的框架,以便据此实施于软件设计和构造中.

(5) 软件体系结构的演化和扩展(Software Architecture Evolution and Extension).在实施软件体系结构时,根据系统的需求,常常是非功能需求,如性能、容错、安全性、互操作性、自适应性等非功能性质影响软件体系结构的扩展和改动,这称为软件体系结构的演化.由于对软件体系结构的演化常常由非功能性质的非形式化需求描述引起,因而需要重复第 1 步,如果由于功能和非功能性质对以前的软件体系结构进行演化,就要涉及软件体系结构的理解,需要进行软件体系结构的逆向工程和再造工程.

(6) 软件体系结构的提供、评价和度量(Software Architecture Provision, Evaluation and Metric).这一阶段通过将软件体系结构实施于系统设计后,系统实际的运行情况,对软件体系结构进行定性的评价和定量的度量,以利于对软件体系结构的重用,并取得经验教训.

(7) 软件体系结构的终结(Software Architecture Termination).如果一个软件系统的软件体系结构进行多次演化和修改,软件体系结构已变得难以理解,更重要的是不能达到系统设计的要求,不能适应系统的发展.这时,对该软件体系结构的再造工程即不必要、也不可行,说明该软件体系结构已经过时,应该摈弃,以全新的满足系统设计要求的软件体系结构取而代之.

本文主要涉及软件体系结构生命周期模型的规范描述,对其他方面的研究将另文详述.

2　软件体系结构的分类和建模

2.1　软件体系结构的分类

总的说来,软件体系结构可分为两大部分:一般的、抽象的软件体系结构和特定域的软件体系结构.

对于抽象的软件体系结构的风格和模式,文献[4]对其进行了分类.主要有管道-过滤器.面向对象组织,基于事件的、隐式激发机制的系统,仓储,分层系统,表驱动的解释器,分布式进程,主程序/子程序组织的系统,状态转移系统,进程控制系统等.在此,主要介绍管道-过滤器软件体系结构,以备下文讨论.在管道-过滤器的软件体系结构中,每个构件有一组输入和输出.构件从输入中读出数据流,在输出口产生数据流.一般地,构件对输入流作局部转换并不断计算,以使输出和输入可并发执行.连接件作为流的导管将一个构件的输出传送到另一个构件的输入.例如:UNIX 的 Shell 和编译器的管道线结构.

特定域的软件体系结构可以有很多.文献[5]介绍了军事领域的软件体系结构,包括:智能武器、军事命令与控制系统、制造执行系统、自动车辆管理系统等.在电信领域,随着技术的发展,出现了智能网、电信管理网和电信信息网络体系结构等软件体系结构.

2.2 软件体系结构的建模

研究软件体系结构的首要问题是如何表示软件体系结构,即如何对软件体系结构建模.根据建模的侧重点的不同,可以将软件体系结构的模型分为5种:结构模型、框架模型、动态模型、过程模型和功能模型.[6]在这5个模型中,最常用的是结构模型和动态模型.

· 结构模型:这是一个最直观、最普遍的建模方法.这种方法以体系结构的构件、连接件和其他概念来刻画结构,并力图通过结构来反映系统的重要语义内容,包括系统的配置、约束、隐含的假设条件、风格、性质.研究结构模型的核心是体系结构描述语言.

· 框架模型:框架模型与结构模型类似,但它不太侧重描述结构的细节而更侧重于整体的结构.框架模型主要以一些特殊的问题为目标建立只针对和适应该问题的结构.

· 动态模型:动态模型是对结构或框架模型的补充.研究系统的"大颗粒"的行为性质.例如,描述系统的重新配置或演化.动态可能指系统总体结构的配置、建立或拆除通信通道或计算的过程.这类系统常是激励型的.

· 过程模型:过程模型研究构造系统的步骤和过程.因而结构是遵循某些过程脚本的结果.

· 功能模型:该模型认为体系结构是由一组功能构件按层次组成,下层向上层提供服务.它可以看作是一种特殊的框架模型.

这5种模型各有所长,也许将5种模型有机地统一在一起,形成一个完整的模型来刻画软件体系结构更合适.文献[7]提出了一个4+1的视角模型.4+1模型从5个不同的视角包括逻辑视角、过程视角、物理视角、开发视角和场景视角来描述软件体系结构.每一个视角只关心系统的一个侧面,5个视角结合在一起才能够反映系统的软件体系结构的全部内容.

3 软件体系结构建模语言

由于软件体系结构建模语言是描述软件体系结构规范的出发点,是研究软件体系结构的核心,是分析和验证软件体系结构的前提和基础.因此,目前对软件体系结构的研究大都以软件体系结构语言为核心展开研究.

3.1 几个典型的软件体系结构描述语言

在以上所讲的模型中,对结构模型和动态模型研究得最多.许多体系结构描述语言ADL(architecture description language)都支持对结构模型和动态模型的描述.然而,尽管所有的ADL都对结构进行研究,但它们却在研究范围上有着很大的区别.大体上来说,可以将其归纳为3类.

3.1.1 研究软件体系结构配置结构的描述语言

这一类软件体系结构描述语言主要针对体系结构的静态和动态配置,对体系结构配置的演化所具有的性质进行研究.典型的体系结构描述语言有Darwin[8]和CHAM(chemical abstract machine).[9]Darwin用π演算来对系统的行为进行建模.该模型具有描述高度动态特征的体系结构的灵活性,即构件和连接件的组织结构可以在系统运行时动态改变.利用π演算的强类型系统能进行静态检查.CHAM可通过对处理单元、数据单元和连接单元的描述、引入的转换规则以及项重写来描述和分析体系结构的动态行为,它实质上是一种结构的重写系统.

3.1.2 研究软件体系结构实例的描述语言

这一类体系结构描述语言描述的是特定的系统,它所解决的问题是:"系统的体系结构是什么".例如:Rapide[10]就属于这类体系结构描述语言.它是一个基于事件的、并发的面向对象的语言,专门为系统体系结构建立快速原型而设计.它由5个子语言组成:描述构件接口的类型语言、描述构件间事件流向的体系结构语言、限制构件行为的抽象的规范语言、编写可执行模块而用的可执行语言以及描述事件模式的模式语言.通过这5种语言的结合使用,使得系统在确定实现策略之前首先在体系结构的层次上进行分析和设计.

3.1.3 研究软件体系结构风格的描述语言

这一类语言描述体系结构的模式或范型,从而描述了具有相同风格的一族系统的体系结构,它所解决的问题是"系统使用的结构模式"和"体系结构风格的含义".典型的体系结构描述语言有Wright.本文将重点研究这类软件体系结构描述语言.在Wright[11]中,最重要的是连接件.连接件由一组角色和胶水规格组成.角色定义了每个交互方所期望的本地行为.胶水规格则描述了各个角色如何协调在一起.Wright采用CSP(communicating sequential

process)[12]来描述.而文献[13]采用 Z[14]语言来描述软件体系结构的风格.该方法用一组从语法描述域到软件体系结构风格的语义域的语义函数映射来描述软件体系结构.

从文献[13,14]描述的连接件中可以看出,CSP 适合描述行为,因而能描述和分析软件体系结构的动态行为特性.文献[11]用 CSP 分析了软件体系结构连接件的某些重要性质,如在一定条件下满足无死锁性.而 Z 语言是基于模型和集合论的规范语言,它类似于指称语义,适合描述软件体系结构的某些静态性质,如风格、子风格的定义.[13]但它却难以描述和分析软件体系结构的动态性质.本文的目标就是寻求一种规范的形式化方法,使它既能描述和分析软件体系结构的静态语义,又能描述和分析软件体系结构的动态语义,同时,不仅能描述和分析无死锁性等安全性,而且还能描述和分析活性.而时序逻辑语言 XYZ/E[15]正符合这一要求,这是由于 XYZ/E 是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,由于时序逻辑建立在一阶谓词逻辑的基础上,它又引入了时序逻辑关系来表达自动机的状态迁移,因而结合了静态语义和动态语义的特点,适应软件工程的许多领域,包括逐步求精的设计方法;速成原型法;抽象描述和验证;分布式和基于共享存储的并发程序设计;从非形式化描述到形式化描述的平稳过渡.最后,由于它建立在时序逻辑的基础上,对于描述和分析静态和动态性质具有很强的能力.因此,XYZ/E 非常适合于描述软件体系结构.

3.2　软件体系结构描述语言 XYZ/SAE

基于以上的分析,我们提出一种用时序逻辑来描述软件体系结构的建模语言 XYZ/SAE(XYZ/E for software architecture),它应该是时序逻辑语言族 XYZ/E(XYZ/element)的一个子语言.如果从前面对软件体系结构语言的分类来说,XYZ/SAE属于一种研究软件体系结构风格的描述语言.本文采用时序逻辑语言族 XYZ/E 来描述软件体系结构,目的就不仅仅在于对软件体系结构给予精确的描述规范,更重要的是对软件体系结构的性质进行分析以及对软件体系结构的求精、验证和演化进行分析.因而力求将软件体系结构的规范描述、逐步求精以及验证和演化统一在时序逻辑框架内.本章主要针对软件体系结构的规范描述进行研究,在以后的章节里,则着重讨论软件体系结构生命周期的其他阶段.

我们认为通过将抽象的软件体系结构逐步求精,直至具体的软件体系结构可以达到软件的半自动生成和自动生成;可以采用统一的时序逻辑来描述和验证不同抽象级的体系结构的一致性;软件体系结构在高层上是抽象描述,而实际上随着软件体系结构的逐步求精,最后将变成可执行的软件.由于 XYZ/E 能在统一的时序逻辑框架下同时支持抽象描述和可执行程序,因此,XYZ/E 恰好能满足这一要求.另外,由于时序逻辑的特点,对于分析软件体系结构的演化等动态行为等具有很强的分析能力.

XYZ/SAE 软件体系结构描述语言框架

(1) XYZ/SAE 语言的基本语句和基本程序

XYZ/SAE 的基本语句和基本程序单元即为 XYZ/E[15]的所有基本的简单语句和基本的程序单元.其基本语句以逻辑型合式公式的规范形式定义,包括经典的一阶逻辑的连接词、量词和时序逻辑中的时序算子.经典的一阶逻辑连接词、量词有:~(非)、∧(与)、$V(或)、⇒(蕴涵)、==(等于)、$A(全程量词)、$E(存在量词)、$T(真)、$F(假).时序算子有:◇(最终算子)、□(一直)、$O(下一时刻)、$U(直到算子)、$W(除非)及一些不常用的过去时序算子.其基本语句参见文献[15].

(2) XYZ/SAE 的程序构造单元和并发机制

XYZ/SAE 还包括包块(Package 用%PACK 定义)、过程(Procedure 用%PROC 定义)、进程(Process 用%PROS 定义)以及表示并发性的不确定性选择语句(用符号!! 表示)和并行语句(用符号 ‖ 表示).其定义和语义参见文献[15].

(3) XYZ/SAE 的框架构造形式

本文认为,软件体系结构的描述可分为以下几步:

第 1 步,也是最重要的步骤是先描述连接件,因为连接件将构件有机地联系在一起,决定了软件体系结构的风格.连接件的描述可分为对参与连接件的各个角色的描述及用一组时序逻辑合式公式表达的逻辑限制和制约条件.角色是构件在连接件中进行交互的抽象,其描述包括静态功能和动态的行为描述.连接件的动态描述为各个角色的并发执行.而时序逻辑合式公式表达的限制制约条件实际上与文献[11]中的胶水概念相类似.

```
ConnDeclPart::=%CONN  ConnectorName==[[RoleDeclPart;]
                                      Probody]
                                      [WherePart]
RoleDeclPart::=%ROLE[Role,...,Role]
```

Role::=[Package,Process]
WherePart=Glue

上式说明,连接件的声明部分 ConnDeclPart 可分为3部分.角色声明部分 RoleDeclPart、连接件角色参与交互的程序体 Probody 及对连接件及其角色的约束限制条件 WherePart.其中角色声明 RoleDeclPart 可声明多个角色.每个角色由描述角色静态条件的 Package 及描述角色动态条件的 Process 组成.

第2步可以通过实例化连接件中的角色为软件体系结构的构件,从而实例化软件体系结构本身.同时可进一步描述该实例化的软件体系结构的配置拓扑结构等其他约束限制.

Porgram::=%SAPROG ProgramName==[[CompDeclPart;]
Probody]
[WherePart]

一个描述软件体系结构的程序可由3部分组成:构件声明部分 ComDeclPart、构件交互的程序体 Probody 及各构件的约束限制条件.其中构件对连接件中的角色进行实例化.

构件是软件体系结构中显式地独立存在的实体或对象,构件声明与角色的声明相同,只是它可根据其功能,实例化角色中的操作.

4 典型的软件体系结构描述

下面以管道和过滤器体系结构为例,来说明体系结构描述语言 XYZ/SAE.

```
%ROLE[Source:
  %PACK[source:
    %VAR[sourcedata:DATA]
    %PROC[
      delivedata(%IOP deliver:seq DATA)==[
      %ALG[
      LB=START_delivedata∧(sourcedata≠NULL)∧(deliver==NULL) /* pre condition */
        ⇒◇(sourcedata==NULL)∧(deliver≠NULL) /* post condition */
      ]∧ $OLB=RETURN]]
    WHERE □(strcat($Osourcedata,deliver)==sourredata)
  %PROS[
    write(CHN source-to-sink(*,sink):seq DATA)==[
        %LOC[stream:DATA,notification:STRING,s1,s2:INT]
        %ALG[/* behaviour description */
          LB=START_Source⇒(s1=0)∧(s2=0)∧ $OLB=l0;
          LB=l0⇒ $Osource-to-sink?notification∧ $OLB=l1;
          LB=l1⇒(!![(notification==“write”)▷(delivedata(IOP deliver]stream)∧(s1=1)∧ $OLB=EXIT,(notification==“close”)▷(s2=1) ∧ $OLB=EXIT];
          LB=l2∧(s1=1)⇒ $Os1=0∧ $OLB=l0;
          LB=l2∧(s2=1)⇒ $Os2=0∧ $OLB=STOP;]
  ];
]/* end of ROLE Source */
```

角色 Source 包括一个包块和进程 write,其中包块将源数据与传送数据的操作封装在一起.在传送开始时,源数据不空,传送的数据为空.在传送结束时,源数据为空,传送数据不空.在整个传送过程中,下一时刻的源数据加上传送的数据等于该时刻的源数据.进程 write 负责和角色 Sink 的 Read 通信,直到管道关闭为止.同理可描述角色 Sink.

```
%ROLE [Sink:
  %PACK[sink:
    %VAR[sinkdata:DATA]
    %PROC[
      receiveda·a(%IOP receiver:scq DATA)==[
      %ALG[
      LB=START_receivedata∧(sinkdata==NULL)∧(receiver≠NULL)/* pre condition */
        ⇒◇(sinkdata≠NULL)∧(receiver==NULL) /* post condition */
      ]∧ $OLB=RETURN]]
    WHERE □( $Osinkdata==strcat(sinkdata,receiver))
  %PROS[
```

```
        read(CHN source-to-sink(writer, *),seq DATA)==[
            %LOC[stream:DATA,notification:STRING,s1,s2:INT]
            %ALG[ /* behaviour description */
                LB=START-Sink→(s1=0)∧(s2=0)∧ $OLB=10;
                LB=10⇒ $Osource-to-sink?notification∧ $OLB=11;
                 LB=11⇒(!![((notification==“read”) $V(notification==“read-eof”))]▷(s1=1)∧ $OLB=EXIT,(notification==“close”)]▷(s2=1)∧ $OLB=EXIT];
                LB=12∧(s1=1)∧(notification==“read”)⇒ $Os1=0∧receivedata(IOP receiver|stream)∧ $OLB=10;
                LB=12∧(s2=1)∧(notification==“read-eof”)⇒ $Os2=0∧ $Osource-to-sink?notification∧ $OLB=13;]
                LB=13∧(notification==“close”)⇒ $OLB=STOP]
            ];
    ]/* end of ROLE Sink */
    %CONN Pipe==[
      ... /* declaration of Role Source and Sink */
      LB=START⇒ $Osource==Source∧ $Osink==Sink∧ $OLB=10;
      LB=10→‖[source,sink];
    ]
    WHERE Glue
```

其中 Glue 对管道 Pipe 的制约限制为：Pipe 传输之前要保证管道为空，否则传送了一些不该传递的数据流. 传输过程中不仅要保持数据流的先进先出的顺序，而且数据流不能丢失；只要管道能够写或读，所有写入管道的数据流最终将必然要写入 Pipe 或从 Pipe 中读出. 前一性质是安全性，后一性质是活性.

$$
\begin{aligned}
Glue \equiv & ((start(pipe) \rightarrow empty(pipe)) \wedge \\
& \square(write(pipe) \rightarrow \$Odeliver = deliver \wedge \$Opipe = enter(pipe, deliver)) \wedge \\
& \square(read(pipe) \rightarrow \$Oreceiver = head(pipe) \wedge \$Opipe = tail(pipe)) \wedge \\
& \square(close(pipe) \rightarrow sum(deliver) = sum(receiver)) \wedge \\
& (writeenabled(pipe) \rightarrow \Diamond write(pipe)) \wedge \\
& (readenabled(pipe) \rightarrow \Diamond read(pipe)))
\end{aligned}
$$

对于文献[4]中提到的限制管道上可驻留的数据流量的 bounded pipe 以及限制两个过滤器之间传递的数据，要有良定义类型的 typed pipe，可通过添加若干合式公式很容易地得到. 因而可以很容易地定义软件体系结构的风格和子风格. 例如：对于 bounded pipe 可加入合式公式：$(\sim full(pipe) \rightarrow writeenabled(pipe)) \wedge (\sim empty(pipe) \rightarrow readenabled(pipe))$.

定义3. 若连接件 Connector1和 Connector2的胶水分别为 Glue1和 Glue2，如果 Glue1⇒Glue2，则称连接件 Connector2所代表的软件体系结构风格为 Connector1所代表的软件体系结构风格的子风格.

一个具有 n 个过滤器的拓扑结构为线性的管道-过滤器软件体系结构为：

```
%SAPROG Pipeline-Filter==[ /* 可执行的体系结构动态模型 */
    ...
  LB2=10⇒ $Ofilters(1)==pipe1(Source)
          ∧ $Ofilters(2)==(pipe1(Sink) $V pipe2(Source))
          ∧...
          ∧ $Ofilters(i)==(pipei-1(Sink) $V pipei(source))
          ∧...
          ∧ $Ofilters(n)==pipen-1(Source);
  LB2=11⇒‖[filters(1),filters(2),...,filters(r)]
  ]
WHERE □($A i,i∈1,...,n-1:pipei∈(filters(i),filters(i+1))∧pipei⇒Glue)
```

一个有 n 个过滤器、拓扑结构为线性的软件体系结构 Pipe-Filter 为各个过滤器的并发执行，同时各个连接件 $pipe_i$ 满足 Pipe 的描述.

从上例中可以看到，软件体系结构的动态行为通过 XYZ/SAE 得到了很好的描述. 同时，由于 Package 及 WherePart 部分的加入，可以描述软件体系结构的静态行为. 它还可以很容易地定义和描述软件体系结构的风格和子风格，从而吸取了 CSP 和 Z 描述的优点，弥补了它们各自的不足. 更重要的是，不像 Wright 和 Z 语言仅仅是一种说明陈述语言，XYZ/SAE 既是逻辑系统，又是可执行的语言，从而便于动态仿真和验证，避免了 Rapide 中可执行语言需要另一子语言的做法.

5 结束语

本文提出的软件体系结构描述语言XYZ/SAE能同时描述软件体系结构的静态和动态语义,避免了CSP和Z语言中只能描述其中一种的不足,同时它又是一个可执行的体系结构描述语言,可以描述和分析软件体系结构设计中的安全性和活性,避免了Rapide中可执行语言需要另一子语言的做法.今后的工作是建立XYZ/SAE及可视化工具,并支持软件体系结构生命周期的各部分集成到XYZ系统中.

致谢 感谢唐稚松院士在北京邮电大学所作的关于XYZ/E的精彩演讲,本文的许多思想得益于唐先生的启发,在此向唐先生表示衷心的感谢.

参考文献

1 Perry D E, Wolf A L. Foundations for the study of software architecture. ACM SIGSOFT Software Engineering Notes, Oct. 1992, 17(4):40~52
2 Garlan D, Perry D E. Introduction to the special issue on software architecture. IEEE Transactions on Software Engineering, April 1995,21:269~274
3 Shaw M, Garlan D. Software Architecture: Perspectives on an Emerging Discipline. Englewood Cliffs: Prentice Hall, 1996
4 Garlan D, Shaw M. An introduction to software architecture. In: Advances in Software Engineering and Knowledge Engineering, Vol 1. New York: World Scientific Publishing Company, 1993
5 Terry A *et al*. Overview of Teknowledge's domain-specific software architecture program. ACM SIGSOFT Software Engineering Notes, Oct. 1994,19(4):68~76
6 Garlan D, Paulisch F N, Tichy W F. Summary of the dagstuhl workshop on software architecture. ACM Software Engineering Notes, July 1995,20(3):63~83
7 Kruchten P B. The 4+1 view model of architecture. IEEE Software, November 1995,12(6):42~50
8 Magee J, Kramer J. Dynamic structure in software architectures. ACM SIGSOFT Software Engineering Notes, Nov. 1996,21(6):3~14
9 Inverardi P, Wolf A. Formal specification and analysis of software architectures using the chemical abstract machine model. IEEE Transactions on Software Engineering, 1995,21(4(special issue on software architecture)):373~386
10 Luckham D C, Augustin L M *et al*. Specification and analysis of system architecture using Rapide. IEEE Transactions on Software Engineering, 1995,21(4(special issue on software architecture)):336~355
11 Allen R, Garlan D. Formalizing architectural connection. In: Proceedings of the 16th International Conference on Software Engineering. May 1994. 71~80
12 Hoare C A R(著),周巢尘(译). 通信顺序进程. 北京:北京大学出版社,1990
(Hoare C A R. Communicating Sequential Process. Englewood Cliffs: Prentice-Hall, 1985)
13 Abowd G, Allen R, Garlan D. Using style to understand descriptions of software architecture. ACM SIGSOFT Software Engineering Notes. Dec. 1993,18(5):9~20
14 Sprivey J. The Z Notation: A Reference Manual. Englewood Cliffs: Prentice Hall, 1989
15 唐稚松,赵琛. 一种面向软件工程的时序逻辑语言. 软件学报,1994,5(12):1~16
(Tang Zhi-song, Zhao Chen. A temporal logic language oriented toward software engineering. Journal of Software, 1994,5(12):1~16)

A Study of Software Architecture Modeling

ZHOU Ying-xin AI Bo

(*College of Computer Science and Technology Beijing University of Posts and Telecommunications Beijing* 100088)

Abstract The authors promote the concept of software architecture engineering, establish the life cycle of software architecture, classify the different kinds of software architectures and architecture modeling, introduce several typical software architecture modeling languages, give a formal architectural style modeling language XYZ/SAE(as a sub-language of a series of temporl logic languages——XYZ/E) in this paper. It supports the description of component, connector and configuration, and also static and dynamic aspects of software architecture.

Key words Software architecture, software architecture modeling, software architecture modeling language, temporal logic, component, connector.