基于主动连接件的软件体系结构及其描述方法*

张家晨 冯铁 陈伟 金淳兆

(吉林大学计算机科学系 长春 130023)

E-mail: selab@mail.jlu.edu.cn

摘要 连接件是软件体系结构描述中的重要概念.该文在连接件中引入引擎机制,使其在与相关组件连接的过程中成为运行活动的触发元素,而且在目标系统实现中转化为可见的组成部分,这种连接件称作主动连接件.文章介绍一种基于主动连接件的软件体系结构的基本概念及其描述语言.

关键词 软件体系结构,面向对象方法,软件体系结构描述语言,组件,主动连接件.

中图法分类号 TP311

在软件复用技术的研究和应用过程中,人们越来越重视软件的高层结构,以使这种结构能应用于不同的软件开发之中,从而提高软件复用的粒度.软件体系结构(software architecture,简称SA)的研究便是在这种目的的驱动下产生和发展的.如何描述SA是以SA的软件开发技术为基础的.目前,关于SA的描述,出现了一系列方法和相关的描述语言(architecture description language,简称ADL).主要有ACME,Unicon,Wright,Asope,Rapide,Armani,C2,SADL[1~4]等.在这些ADL中,组件(component)是完成特定功能的计算场所,体现为相对独立的编译单位,它通过端口所提供的接口与外界相联系;连接件(connector)用来表示组件之间的联系,相关组件通过它连接起来形成系统.与传统的移入/移出(import/export)式的模块接口语言相比,这些ADL强调连接件的作用.在SA的描述中,组件和连接件具有同等的地位,一般都描述它们的接口、类型、语义、限制及细化问题[5,6].

在系统实现的目标代码中如何实现连接件,对于系统设计的可跟踪性、软件的自动化和逆向工程具有重要意义.目前,连接件在系统实现中的转化形式包括:转化成一个计算的实体(与组件相同)、成为共享的数据存储单位(如变元或缓冲区等)、分化融合到相关组件的实现中变得不可见、体现为一些消息通信机制或协议等[7,8].这种连接件在目标系统转化中的不确定表现,尤其是不可见的情形,对系统的可跟踪性影响较大,因而不利于软件自动化和逆向工程.解决这个问题的一种途径是对连接件的作用进行扩充,使它在最终实现系统中转化成相对独立的一段代码,从而是可见的.本文提出了一种基于主动连接件的软件体系结构ACSA(active-connectors-based software architecture),基于这种思想,定义了软件体系结构描述语言Tracer,并研制了支持软件体系结构辅助开发的工具原型.

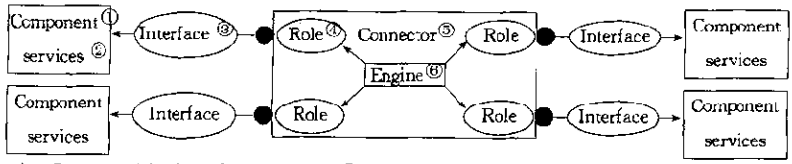
1 ACSA的基本概念

ACSA的基础是面向对象方法,它的基本元素组件、组件的接口、连接件及连接件的角色都是对象.如图1所示,组件通过接口与连接件装配构成软件的体系结构.下面,我们对ACSA中的基本元素进行介绍.

* 本文研究得到国家自然科学基金(No.69773044)、国家"九五"重点科技攻关项目基金(No.98-780-01-07-07)和吉林大学青年基金(No.1999A506)资助.作者张家晨,1969年生,副教授,主要研究领域为软件复用技术,软件体系结构描述方法,软件自动化方法.冯铁,1972年生,讲师,主要研究领域为软件复用技术.陈伟,1969年生,讲师,主要研究领域为面向对象方法,软件测试方法.金淳兆,1937年生,教授,博士生导师,主要研究领域为软件工程.

本文通讯联系人:张家晨,长春 130023,吉林大学(北区)计算机科学系

本文2000-01-17收到原稿,2000-04-10收到修改稿



$A \rightarrow B$ means A is dependent on B, A—● B means the type of A is fit for the type of B.
$A \rightarrow B$表示A依赖于B, A—● B表示A的类型适应于B的类型.
①组件,②服务集,③接口,④角色,⑤连接件,⑥引擎.

Fig. 1　Active connector in the configuration of components
图1　主动连接件结构及组件的装配图

1.1 组　件

组件是相对独立的功能计算单位.虽然采用不同的软件开发方法,在不同的软件抽象层次上组件的组成可能不同,但在描述组件与相关元素之间的关系时,可以把组件看作对象.组件由组件类定义,主要关注的是它能提供的服务集合.组件的服务集反映了组件的抽象数据类型.组件的计算和数据通过其服务集来访问.

组件分为两类,一类是简单组件,其服务集中的服务都已经实现,或者可以通过继承的方式得以细化实现.这种组件在进行软件体系结构描述时,不需要细化描述;另一类是复合组件,复合组件是为描述和实现软件系统的高层次组织关系而引入的,其服务集中的服务一般需要若干抽象度相对较低的组件来细化描述.这些抽象度相对较低的组件及其关系又构成了抽象级别较低的体系结构.复合组件的服务集依赖于它所包含的组件的服务集.

1.2 连接件

人们在对软件系统中的组件进行结构上的安排时,形成了一些典型的组织方式或模式,如将若干组件组成管道-过滤器模式或黑板模式等.对于这些组织方式,通常用连接件定义.连接件是应用于组件间进行连接通信的设计框架,它的领域无关性类似于设计模式(design patterns)[9].

连接件由连接件类定义.一个连接件由若干角色(role)和一个引擎(engine)组成.角色是具有自身特定功能和作用的不依赖于任何应用情形的对象,是对能够连接到连接件上、在连接中处于相同地位的所有组件的抽象.它的类型也由类定义.每个角色由相应的具体组件通过装配充当替换.

连接件中角色之间有静态和动态两种关系.静态关系由各角色的类型定义以及它们之间的继承、聚合和关联关系来描述.角色之间的动态关系由引擎描述,引擎描述了角色之间的联系、协同工作等动态行为,它定义了所有角色参与连接的活动过程.引擎可以从某些角色中获得信息,也可以将获取的信息发送给某些角色.在这种关系中,所有角色是不直接相关的,它们只能被引擎使用,相互之间的通信也是通过引擎来定义完成.引擎与角色之间的这种主动和被动的关系,决定了装配后的连接件与相关组件之间是主动和被动的关系,即在实现系统运行过程中,相关组件的连接活动是由连接件中的引擎驱动完成的.这样的连接件称作主动连接件.

1.3 组件的装配和组件接口的制作

组件装配时建立软件体系结构的重要活动是将组件实例连接到相应的连接件实例上.一个组件能与一个连接件进行连接的前提是:它的类型必须适应于连接件中相应的角色类型.一般来说,利于复用的组件在研制时关心的是它的功能,并不一定考虑到将与什么样的连接件相连接,组件的服务集所构成的抽象数据类型往往不能适应于角色的类型要求.所以,为了满足连接条件,当把组件装配到连接件上时,需要对组件进行一定的包装,对组件进行包装后的结果称作组件的接口.组件的接口也是对象,它与组件的关系是单向的依赖关系.如图1所示,一个组件与其他组件相关联是通过其接口经过连接件连接完成的.一个组件可以有多个接口,每一个接口可以扮演连接件中所对应的角色.由于组件接口类型和角色类型都是类,所以在制作接口时,可以把接口的类型制作成角色类型的子类型,从而满足类型适应的约束.这里,子类型的定义是:若类型A提供了类型B的所有操作,则称类型A是类型B的子类型.

根据要扮演的角色类型要求,设计组件接口的一般方法是:以该角色类型为基类,派生得到一个新的类作为接口类,并且新类中有一个对组件的允引.接口类中的服务利用组件中的服务实现.在组件装配之后,连接件引

擎代码中使用的角色实际上是组件接口的实例.

1.4 软件体系结构的描述

软件系统的体系结构描述的主要内容是软件系统中的组件及其关系.组件间的关系从3个角度描述,即组件之间的静态关系、组件之间的动态关系和组件的物理分布信息.

组件之间的静态关系和动态关系是通过连接件的类型定义和组件装配确定的.在一个系统中,如果把组件A与组件B装配到连接件L上,则意味着组件A与组件B拥有了L中定义的相关角色之间的静态关系和动态关系.

组件的物理信息是对系统实现的要求.这些信息包括组件对硬件的要求、组件之间的关联对地理上的要求、远程通信对时间上的要求以及对通信协议的要求等.

由于连接件类型和它的角色类型都是由类定义的,所以,可以通过继承机制来完成对连接件的细化描述,并且在细化过程中能够加入一些物理信息限制,例如,可以将某一角色类型分成本地的和远程的.引擎的描述或实现也可以通过复写进行相应的细化.

2 基于主动连接件的软件体系结构描述语言Tracer简介

Tracer中存在若干预定义类型,主要的有容器类Collection和公共对象类Object,公共对象类是所有数据类型的基类.

2.1 组件的描述

```
ComponentType〈组件类型名〉
    Services〈服务集合〉EndServices
    〈实现平台约束描述〉// 组件的物理信息描述.
EndComponentType
```

2.2 连接件描述

```
ConnectorType〈连接件类型名〉
    〈角色类型定义序列〉//详见下面〈角色类型定义〉
    EngineCode
        〈过程式描述〉//在Tracer中,用自定义的扩展C++语言LikeC++描述,主要扩展的机制是并
                发控制和远程过程调用
    EndEngineCode
EndConnectorType
〈角色类型定义〉::=RoleType〈角色类型名〉
        ClassMethods
            〈类方法说明序列〉      // 获取实例的方法
        VirtualOperators
            〈操作说明序列〉        // 每一个操作必须被待匹配的组件接口复写
        ImplementedOperators
            [〈操作说明序列〉]      // 每个操作已经实现,不必被复写,可缺省
EndRoleType
```

2.3 软件系统的体系结构描述

```
System〈系统名〉
    [〈组件类型描述序列〉]
    [〈连接件类型描述序列〉]
    〈组件实例说明序列〉        // 用组件类型说明所要用到的组件实例
    〈连接件实例说明序列〉      // 用连接件类型说明所要用到的连接件实例
```

```
〈组件装配描述序列〉          // 详细见下面〈组件装配描述〉
EndSystem
〈组件装配描述〉::=〈组件实例〉〈连接操作符〉〈连接件实例〉as a 〈角色类型〉
〈连接操作符〉::=link to                  // 不关心地理信息
              |local link to             // 明确指明是本地连接
              |remote link to [by 〈通信协议名称〉]
                                         // 通过某种通信协议进行远程通信
```

3 ACSA 的辅助工具原型

ACSA 的辅助工具原型包括 ADL 编辑器和语法检查器、类图和组件拓扑关系图形编辑器、组件库、连接件类型库和组件适配辅助工具.

ADL 编辑器和语法检查器用于完成对 Tracer 语言的处理.图形编辑器完成对连接件中角色类型的静态关系描述,系统的拓扑结构也在图形编辑器中显示.同时,由于存在着系统结构图和 Tracer 中关于组件装配描述之间的转化器,组件的装配工作也可以通过图形编辑器来直观地完成.组件适配辅助工具的作用是:完成组件接口的部分代码的自动生成;辅助完成组件接口需要复写操作的手工生成;根据组件的装配描述,生成适应于连接件角色类型的组件接口的实例,并将它们与连接件实例中的角色实例联系起来,从而完成连接件引擎代码中关于角色实例的初始化.

4 举 例

本节定义了一个管道-过滤器类型的连接件,并给出了通过这种连接件进行连接的一个小系统的描述.

(1) 管道-过滤器类型的连接件类型定义

```
ConnectorType MyPipe_Filter
    RoleType    FirstStage
        ClassMethods
            Collection CreateInstances();   //类工厂方法,用于产生角色实例,每个角色类都有
                                            //实例将根据装配信息获取并加入到容器中
        VirtualOperators
            DataType: OutputData;
    EndRoleType
    RoleType    SecondStage
        ClassMethods
            Collection CreateInstances();
        VirtualOperators
            InputData(DataType);
    EndRoleType
    EngineCode
        Collection m_Buffer;
        F=FirstStage::CreateInstances().getFromFirst(); //此例中仅一个实例
        S=SecondStage::CreateInstances().getFromFirst(); //其他角色类型可能有多个实例
        CoBegin
            Repeat
                m_buffer.setToLast(F.OutputData());
            EndRepeat
            Repeat
```

```
                S.Input(m_buffer.getFromFirst());
            EndRepeat
        CoEnd
    EndEngineCode
EndConnectorType
```

(2) 假设有两个待通过该类连接件进行连接的组件,其类型描述分别如下所示:

```
ComponentType ComponentA
    Services
        DataType: getData
        ......
    EndServices
    ......
EndComponent
```

```
ComponentType ComponentB
    Services
        void setData(DataType)
        ......
    EndServices
    ......
EndComponent
```

(3) 装配描述如下:

```
System MiniSystem
    ComponentA comA;        ComponentB comB;
    MyPipe_Filter pf;
    ComA link to pf as a FirstStage;comB link to pf as a SecondStage;
EndSystem
```

(4) 由辅助工具根据装配描述制作的组件接口描述如下:

```
Interface A_01
    BaseClass MyPipe_Filter::First_Stage
    Methods
        Virtual NewInstance {return * this;}
        Virtual DataType OutputData
         {reuturn anComponetA -> getData
        ();}
    EncMethods
    Attrubutes
        CompontA * anComponentA
EndInterface
```

```
Interface B_01
    BaseClass MyPipe_Filter::Second_Stage
    Methods
        Virtual NewInstance {return * this;}
        Virtual IutputData(DataType aData)
         {anComponentB ->setData(aData)}
    EndMethods
    Attrubutes
        CompontB * anComponentB
EndInterface
```

接口 A_01 实例中的 anComponent$A$ 将实际指向装配描述中的 com$A$;接口 B_01 实例中的 anComponent$B$ 将实际指向装配描述中的 com$B$. 同时,两个接口实例与连接件中对应的角色类型实例相结合,从而完成连接件 pf 中关于角色实例的建立.

5 结束语

为加强连接件在体系结构的细化过程中的可跟踪性,本文对连接件进行了扩充,提出了主动连接件的概念. ACSA 将连接件类型看作是面向对象方法中的类,并提供引擎机制,从而强调连接件在与相关组件进行连接中的地位. 连接件的主动性在于连接件中的引擎是连接活动的触发机制,引擎与各组件之间的关系相当于顾客/服务员之间的关系.

ACSA 的基础是面向对象方法,在体系结构的描述过程中,组件、组件接口、连接件及其角色都被看作对象. 连接件中角色的定义通过类的静态模型描述,而引擎相当于这些类的动态模型,它描述了一个软件系统的局部的动态行为. 连接件的结构类似于设计框架,它与设计框架的不同点在于它的领域无关性. 在接口的制作和组件适配中,辅助工具使用了文献[9]中的若干构造式设计模式. 与设计模式所强调的设计思想重用相比,主动连接

件给出的是一种设计结果,这一点又类似于设计框架.

由于面向对象方法自身强调自底向上的软件开发过程,以面向对象方法为基础的ACSA方法在描述相邻层次之间的衔接描述方面还存在着不足.另外,描述语言Tracer在描述能力上还有待于进一步的提高.

参考文献

1 Shaw M, Garlan D. Software Architecture: Perspectives on an Emerging Discipline. Prentice Hall, Inc., Simon & Schuster Bejing Office, Tsinghua University Press, 1996

2 Garlan D, Monroe B, Wile D. Acme: an interchange language for software architecture. Technical Report, CMU-CS-95-219, Software Engineering Institute, Carnegie Mellon University, 1997

3 Monroe R T. Capturing software architecture design expertise with Armani. Technical Report, CMU-CS-98-163 Version 1.0, Software Engineering Institute, Carnegie Mellon University, 1998

4 Luckham D C, Augustin L M, Kenney J J *et al*. Specification and analysis of system architecture using Rapide. IEEE Transactions on Software Engineering (Special Issue on Software Architecture), 1995,21(4):336~355

5 Ducasse S, Richner T. Executable connectors: towards reusable design elements. In: Jazayeri M, Schauer H eds. Proceedings of the 6th European Software Engineering Conference (ESEC'97). Lecture Notes in Computer Science 1301, Berlin: Springer-Verlag, 1997. 483~499

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

7 Medvidovic N. A classification and comparison framework for software architecture description languages. Technical Report, UCI-ICS-02, Irvine: University of California, 1997

8 Shaw M, Clements P. A field guide to boxology: preliminary classification of architectural styles for software system. In: Penny Storms ed. Proceedings of the 21st Annual International Computer Software and Applications Conference (COMPSAC'97). Los Alamitos, CA: IEEE Press, 1997. 6~13

9 Gamma E, Helm R, Johnson R *et al*. Design Patterns: Elements of Rusable Object-Oriented Software. Reading, MA: Addison-Wesley Publishing Company, Inc., 1995

Active-Connector-Based Software Architecture and Its Description Method

ZHANG Jia-chen FENG Tie CHEN Wei JIN Chun-zhao

(*Department of Computer Science Jilin University Changchun* 130023)

Abstract Connector is an important concept in software architecture. Since an engine mechanism is introduced into connector, connectors become triggers when they are connected with components, and can be transformed into visible parts in implementing system. This kind of connector is called active connector. In this paper, the authors introduce the concept of the software architecture that is based on active connectors, and present a description language.

Key words Software architecture, object-oriented method, software architecture description language, component, active connector.