基于软件体系结构的可复用构件制作和组装*

张世琨，张文娟，常 欣，王立福，杨芙清

(北京大学 计算机科学技术系，北京 100871)

E-mail: zske@cs.pku.edu.cn

http://www.cs.pku.edu.cn

摘要：软件复用和基于构件的软件开发是软件工程化开发和工业化生产的必然趋势，可复用构件的制作和集成组装是其中涉及的两个重要活动.首先分析了现有的构件集成方式所存在的不足，引出了基于体系结构的构件集成；接着对几种不同的体系结构类型进行比较，阐述了它们对构件集成的影响；然后提出了支持接口连接式和插头插座式体系结构类型的构件模型以及基于体系结构的构件集成组装框架，并给出了构件的制作和组装过程；最后对工作进行总结，并指出了进一步的研究方向.

关键词：软件构件；软件体系结构；构件组装

中图法分类号：TP311 **文献标识码**：A

复用是成熟的工程领域的一个基本特征，例如，土木工程、化学工程、计算机硬件工程等，通过大量复用经过实践检验的系统体系结构和标准化的构件，使得对于常规的设计问题都可以直接利用现成的解决方案，避免了系统开发时不断地重复设计，从而可以大幅度地降低开发成本、提高生产效率和产品质量.同样，复用也是软件工程走向成熟的必由之路，将为软件危机的解决提供一条现实可行的途径.

软件复用是指重复使用“为了复用目的而设计的软件”的过程[1].通过软件复用，在应用系统开发中可以充分地利用已有的开发成果，消除了在分析、设计、编码、测试等方面的许多重复劳动，可以提高软件开发的效率；同时，通过复用高质量的已有的开发成果，避免了重新开发可能引入的错误，可以提高软件的质量.因此，软件复用可以大大降低软件开发的费用，并显著地提高生产率和产品质量.

与软件复用相关的两个基本开发活动是面向复用的开发(development for reuse)和基于复用的开发(development with reuse)，前者是生产可复用构件的过程，后者是利用现有的可复用构件生产新系统的过程.可复用构件为有计划地、系统地进行复用提供了手段，是实现软件复用的基石，其生产和使用必须满足两个基本前提，即构件接口的标准化和构件的集成机制.例如，在过程化程序设计中，构件是模块(过程和函数)，集成机制是过程调用；在面向对象程序设计中，构件是对象，集成机制是对象之间的消息通信；CORBA 和 COM 提供了相应的构件接口标准和互操作(即集成)机制.软件复用最终体现为可复用构件通过集成机制组装为完整的系统.

在上面提到的几种构件集成方式中，构件之间的(集成)关系，例如，过程调用、消息连接等，都

* 收稿日期：2001-02-05；修改日期：2001-07-05

基金项目：国家“九五”重点科技攻关资助项目(98 780-01)

作者简介：张世琨(1969－)，男，河北沙河人，博士，副教授，主要研究领域为软件工程，软件体系结构；张文娟(1969－)，女，河北保定人，博士，讲师，主要研究领域为面向对象软件开发，软件复用；常欣(1977－)，男，陕西西安人，硕士，主要研究领域为软件体系结构，分布式对象技术；王立福(1945－)，男，河北保定人，博士，教授，博士生导师，主要研究领域为软件工程，信息安全；杨芙清(1932－)，女，江苏无锡人，教授，博士生导师，中国科学院院士，主要研究领域为软件，软件工程，软件工程环境.

是分散在构件的实现中.这种集成方式明显存在的问题是:(1) 构件在集成时,必须了解各自的实现细节;(2) 构件的集成关系固定在构件的实现中,缺乏灵活性.相对而言,在计算机硬件领域,任何计算机厂商生产的配件,在进行集成组装时,只要遵循标准的接口规范,都可以方便地集成到系统中,而无须了解配件内部的结构和实现细节.这就提醒我们,理想的构件集成方式是,构件接口提供了足够多的用于集成的信息,集成在接口处即可完成.而传统的软件构件接口部分通常只定义了其对外提供的功能,构件对外要求的功能隐藏在构件实现的细节中,因此难以根据接口处的信息定义构件的集成.

在软件体系结构设计层次上,构件通过接口定义了同外界的信息传递以及所承担的系统责任,构件接口包括了构件同周围环境的全部交互内容,也是构件同外界唯一的交互途径.除此之外,环境不应对构件作任何其他与接口无关的假设,例如实现细节等.因此,构件之间的集成可以在接口处定义,从而支持理想的构件集成方式.

本文第1节分析、比较了几种不同的体系结构类型以及它们对构件集成的影响.第2节提出了一个支持在接口处集成的构件模型.第3节讨论了基于体系结构的构件集成框架.第4节对本文的工作进行了总结,并指出了进一步的研究方向.

1 几种不同类型的体系结构比较

一个软件系统的体系结构定义了组成系统的构件与构件之间相互作用的关系[2].体系结构是系统实现的蓝图,为构件的集成组装提供基础和上下文[3].目前对软件体系结构的研究方兴未艾,人们对软件体系结构的认识也不尽统一.Luckham 在文献[4]中分析了3种不同的体系结构类型:对象连接式体系结构(object connection architecture)、接口连接式体系结构(interface connection architecture)和插头插座式体系结构(plug and socket architecture),反映了人们对体系结构的不同认识以及它们在支持构件集成时的不同方式.

1.1 对象连接式体系结构

在这种类型的体系结构中,构件的接口只定义了其对外提供的服务,而没有定义构件对外要求的服务,其中以面向对象中的对象接口为典型代表,所以称这种类型的体系结构为对象连接式体系结构.这种接口定义的非对称性使得构件在集成时,构件对外要求的服务被隐藏在代码的实现细节中,即构件之间的连接关系无法直接在接口处定义,只能是从一个构件的实现到另一个构件的接口.

于是:(1) 当一个构件的接口发生改变时,必须检查系统中每个构件的实现细节,才能发现受此改变而影响到的构件;(2) 即使符合相同接口的构件也不能相互替换,因为新构件未必保持旧构件对其他构件的使用关系.在这种类型的体系结构中,一个构件同其他构件的集成信息被固定在构件的实现中,构件难以适应环境的变化,因而难以复用.究其原因,对象接口对外隐蔽了太多的有用信息,无法满足接口处集成的要求.

1.2 接口连接式体系结构

在这种类型的体系结构中,构件的接口不但定义了其对外提供的功能,而且定义了其要求的外部功能,从而显式地表达了构件对环境的依赖,提高了构件接口规约的表达能力.构件的接口定义了所有对外交互的信息,构件在实现时不是直接使用其他构件提供的功能,而是使用它在接口处定义的对外要求的功能.构件之间的连接是在所要求的功能和所提供的功能之间进行匹配,因此,通

过接口就可以定义系统中构件之间的所有连接.这样,就把上一种体系结构类型中构件之间的固定连接方式变成灵活的连接方式,降低了构件之间的依赖性,提高了构件的独立性和可复用性.

这种类型的构件接口满足通信的完整性,即构件之间的所有通信由接口连接指明.一个构件需要的接口功能 f 和另一个构件提供的接口功能 g 之间的连接,意味着任何对 f 的使用被对 g 的使用所代替.保证了通信的完整性,在前一种类型的构件接口中存在的两个问题——构件接口的改变和符合同一接口的构件替换——也就解决了.

相对于对象连接而言,在构件实现之前就可以建立构件之间的连接.在系统设计时表达了构件接口和集成信息,使得可以根据规约寻找符合要求的构件,或根据需要对外委托加工构件.于是就形成了专业化的分工,即构件生产和系统集成组装,小到一个软件开发组织内部,大到整个软件产业,可以有效地提高系统开发的效率和质量.

1.3 插头插座式体系结构

在接口连接式体系结构中,接口定义满足通信完整性,可以较好地支持接口处的构件集成.但是当接口定义的功能数量很大时,带来了规模上的问题.在接口处标明构件要求的功能更是加剧了这种严峻的形势,而一个功能又往往同几个构件要求的功能连接起来,所以连接的数目也非常大,如同纠缠在一起的一堆乱麻.简单地说,对象连接式体系结构把复杂的连接关系隐藏在构件实现的内部,而当接口连接式体系结构把连接关系显式地标识出来以后,功能连接的数量就成为一个显著的问题.问题的暴露并不是什么坏事,构件之间复杂的连接关系影响到系统的整体结构,需要在系统设计时尽早识别并加以解决.

插头插座式体系结构是接口连接式体系结构的一个特例.为了解决构件接口中的功能和接口连接的规模问题,考虑到构件之间的通信往往涉及到构件接口中功能的成组连接,并且在这组功能之间通常存在着一定的语义约束关系,例如使用顺序和数据流协议等,通过把这样的彼此间关系紧密的功能(包括提供的功能和要求的功能)组织成组,并封装为服务,使得接口中直接包含的内容减少,降低了接口中功能的规模.只有在两个对偶的服务之间才可以连接,对偶的服务是指两个服务所包括的功能完全相同,但其中提供的功能和需要的功能恰好方向相反.这样就进一步降低了接口连接的规模,并且易于检查两个对偶的服务之间的连接的正确性.通过对偶的服务进行连接,是计算机硬件领域中非常普遍的思想,例如串行接口、并行接口、RS232 接口等.

2 构件模型

从以上的讨论可以看出,第 2 种和第 3 种体系结构类型具有较好的性质,表现在构件集成时,不必关心内部的实现细节,支持构件在规约层次上的集成.对于复杂的系统而言,不仅体现在成员构件的数量和复杂的连接关系上,而且成员构件本身通常也是复杂的,是由更简单的低层构件集成而成的,可看做是通常所说的子系统.为了支持系统的这种层次化分解,我们引入了复合构件的概念.系统可被看做是特殊的复合构件,当一个系统集成到更大的系统中时,仅仅是一个普通的复合构件,在这个意义上,我们以统一的观点来看待复合构件和系统.相应地,原子构件是在系统开发中无须再分的最小基本单元,原子构件有其对应的实现体,复合构件在规约层次上表达了成员构件之间的复合,本身并不对应于任何实现体.

下面给出了支持上述第 2 种和第 3 种体系结构类型的构件模型.

构件::=〈构件规约,构件实现〉

构件规约::=〈接口部分,结构部分〉

接口部分::=〈对外提供的功能集合,对外请求的功能集合,服务集合〉

服务::=〈对外提供的功能集合,对外请求的功能集合〉

结构部分::=〈原子构件结构〉|〈复合构件结构〉

原子构件结构::=〈构件实现的引用〉

复合构件结构::=〈引用的构件类型,实例声明,实例连接,映射〉

上面给出的构件模型具有以下性质:

(1) 构件包括规约和实现两个相互分离的部分,系统集成时不必关心构件的实现细节;

(2) 构件规约包括接口部分和结构部分,接口是外部可以唯一对构件所作的假设,包括对外提供的功能、对外请求的功能和服务;

(3) 服务是一组紧密耦合的对外提供和对外请求的功能集合,只有在两个对偶的服务之间才可以连接,对偶的服务是指两个服务所包括的功能完全相同,但其中提供的功能和需要的功能方向恰好相反;

(4) 原子构件的结构部分非常简单,定义了对构件实现的引用,例如实现环境和实现体所在的位置;

(5) 复合构件的结构部分定义了成员构件之间的连接关系,具体包括引用的成员构件的类型、成员构件实例、实例之间的连接关系以及复合构件接口定义向成员构件接口定义的映射.

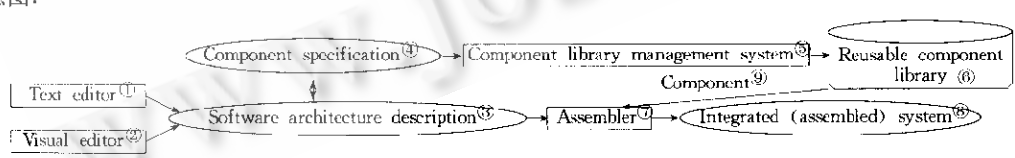
下面是一个简化的编译器 Compiler 的例子,Compiler 是一个复合构件,包括 3 个成员构件:语法分析器 Parser、语义分析器 Semanticizer 和代码生成器 CodeGenerator,Compiler 在规约层次上建立了成员构件之间的连接关系.

```
component Parser is
provides:
    function Initialize();
    function FileName() return String;
requires:
    function Semantize(Tree);
    function Generate(Tree);
end component;
component Semanticizer is
provides:
    function Semantize(Tree);
    function Incremental_Semantize(Context:Tree;Addition:Tree);
requires:
    function FileName() return String;
end component;
component CodeGenerator is
provides:
    function Generate(Tree);
requires:
    function Semantize(Context:Tree;Addition:Tree);
```

```
end component;
component Compiler is
provides:
    function Initialize();
end component;
component body Compiler is
reference:
    Parser,Semanticizer,CodeGenerator;
instance:
    Parser P;
    Semanticizer S;
    CodeGenerator G;
connection:
    P.Semantize to S.Semantize;
    P.Generate to G.Generate;
    S.FileName to P.FileName;
    G.Semantize to S.Incremental_Semantize;
mapping:
    Initialize to P.Initialize;
end component;
```

3 基于体系结构的构件集成框架

人们在进行软件总体设计时,首先根据用户需求和实现环境的要求,确定系统的体系结构,定义系统中的构件与构件之间的关系;在体系结构层次上,构件接口规约对外唯一地代表了构件,故从可复用构件库中查找符合接口规约的构件,必要时开发新的满足要求的构件;最后,通过组装工具把这些构件的实现体组装起来,形成系统.图1给出了基于软件体系结构的构件组装流程的示意图.



①文本编辑器,②可视化编辑器,③软件体系结构描述,④构件规约,⑤构件库管理系统,⑥可复用构件库,⑦组装器,⑧集成组装后的系统,⑨构件.

Fig. 1 Illustration of component assembling based on software architecture

图1 基于软件体系结构的构件组装流程示意图

通过文本编辑器或可视化编辑器可以得到系统的体系结构描述,其中包含了构件接口规约和构件之间连接关系的集成信息.体系结构是在构件接口层次上刻画系统的集成信息,组装器的作用就是把构件接口处的集成映射到构件实现的集成,即把原子构件实现体组装成复合构件或完整的系统.

在统一的体系结构和构件规约描述下,原子构件可以在多种环境中实现.因为目前常见的构件形态,例如C++,Java,CORBA或COM等,并不支持构件在接口层次上的集成,构件的集成信息

隐藏在实现代码中.因此,有必要对构件实现进行约束,使构件的集成信息集中体现在接口层次上.在不同的实现环境下,构件在实现上的约束以及接口集成到实现集成的映射方式也会有所差别,这就导致了不同的组装器.

下面主要以C++语言为例来阐述构件制作和构件组装的基本过程,针对其他实现环境的情况与此类似.

4 构件的制作和组装

基于构件的开发通常包括构件获取、构件分类和检索、构件评估、适应性修改以及将现有构件在新的语境下组装成新的系统.构件获取可以有多种不同的途径:(1) 从现有构件中获得符合要求的构件,直接使用或作适应性修改,得到可复用的构件;(2) 通过逆产工程,将具有潜在复用价值的构件提取出来,得到可复用的构件;(3) 从市场上购买现成的商业构件,即COTS构件;(4) 开发新的符合要求的构件.在进行以上决策时,必须考虑不同方式获取构件的一次性成本和以后的维护成本[5,6].

无论以何种方式获取构件,都必然包含一个构件制作的过程,只有在符合一定规范的构件之间才能进行组装.这里,我们主要以C++作为目标语言,讨论可复用构件的制作和构件组装流程问题.

4.1 原子构件的制作

采用C++语言实现原子构件,可以分为两种情况:(1) 由单个对象封装而成,(2) 更通常的情况,原子构件的粒度比单个对象的粒度要大,是由多个协作的对象封装而成的.

4.1.1 一个原子构件对应单个对象

当一个原子构件的实体对应单个对象时,我们可以称这个构件为单对象构件,其制作流程如下:

(1) 首先定义构件的接口规约部分.

在构件接口规约中定义其对外提供的功能和要求的外部功能.

(2) 制作构件实体,将构件接口规约中定义的对外提供的功能和对外要求的功能,对应到构件的C++实现中.

(a) 建立一个C++的类,类名对应构件名;

(b) 构件对外提供的功能,对应到类的public部分中的函数声明;

(c) 构件对外要求的功能,对应到类的protected部分的一个名为RequiredFunctions的函数数组,每个数组元素存放一个对外要求的功能,并按照规约中描述的顺序在数组中存放.同时,在类的public部分包括一个Initialize函数,负责对RequiredFunctions数组进行初始化;

(d) 实现对外提供的功能对应的各个函数,当涉及到对外要求的功能时,使用RequiredFunctions数组中对应的函数.

4.1.2 一个原子构件对应多个协作的对象

在通常情况下,原子构件的粒度比单个对象的粒度要大,是由多个协作的对象封装而成的,我们称这种构件为多对象构件.

在制作多对象构件时,需要引入一个控制对象,代表原子构件负责对外的交互,并负责其他协

作对象的实例化和建立它们之间的关系.其制作流程如下:

(1) 首先定义构件的接口规约部分.

在构件接口规约中定义其对外提供的功能和要求的外部功能.

(2) 制作构件实体,将构件接口规约中定义的对外提供的功能和对外要求的功能,对应到构件的C++实现中.

(a) 建立一个与构件名同名的C++类,作为控制对象类,负责多对象构件对外的信息通信和对内的协调.控制对象类的声明和实现除了包括上述单对象构件的内容以外,还应包括(2)和(3)的内容.

(b) 在控制对象类的protected部分声明构件中包含的所有其他协作对象,可以是嵌入对象或嵌入指针的方式.如果是嵌入指针的方式,则需要在控制对象类的构造函数和析构函数中分别创建和删除相应的协作对象.

(c) 在控制对象类的Initialize函数中,不仅对其自身的RequiredFunctions数组进行初始化,而且调用各协作对象的Initialize函数,以此建立控制对象和协作对象各自的外部环境,即对外要求的功能.

(d) 实现所有其他协作对象对应的类,类似于上述单对象构件的制作方法.类的protected部分包含一个名为RequiredFunctions的函数数组,用来存放对外要求的各个功能.同时,在类的public部分包括一个Initialize函数,负责对RequiredFunctions数组进行初始化.实现类的各个函数,当涉及到对外要求的功能时,使用RequiredFunctions数组中对应的函数.

4.2 复合构件的制作

复合构件是由一组关系紧密、相互协作的成员构件连接而成的.正如前面讨论的,我们可以用统一的观点来看待复合构件和系统.复合构件的引入为更高层的设计复用提供了可能.其中的成员构件可以是原子构件或复合构件,它们之间的连接的建立就是匹配构件各成员构件对外提供的功能和对外需求的功能,并将复合构件对外提供和要求的功能映射到成员构件相应的功能上去.

复合构件并没有直接对应的实现体,其定义部分除了包含对外提供的功能和要求的功能以外,还包括了其内部成员构件的接口以及它们之间的关系.其制作流程如下:

(1) 首先定义构件的接口规约部分.

在构件接口规约中定义其对外提供的功能和要求的外部功能.

(2) 确定复合构件所包含的成员构件.

(3) 建立复合构件内部的成员构件之间的接口连接关系.

(4) 将复合构件对外提供和要求的功能映射到内部成员构件相应的功能上.

在建立复合构件内部成员构件之间的连接关系以及复合构件和成员构件的功能映射时,应遵循以下规则:

(1) 成员构件对外要求的功能不可以有多于一个的提供者;

(2) 成员构件对外提供的功能可以有多个使用者;

(3) 复合构件对外要求的功能可以映射到多个成员构件的功能;

(4) 复合构件对外提供的功能不可以映射到多于一个成员构件的功能.

4.3 构件组装过程

构件组装的目标是利用现有构件组装成新的系统.我们的方法是将构件在接口处的连接映射

到构件在实现体中的连接.因为系统也可以看做是一个复合构件,而复合构件是可以被逐层分解的,因此,一个典型的复合构件呈现为树型结构.在第4.2节复合构件的规约中,构件对外提供的功能和要求的外部功能分别被映射到成员构件相应的功能上,这就意味着,我们可以把复合构件之间的连接信息逐层进行消解,最终归结为原子构件之间的连接.然后,把原子构件在接口处的连接映射为构件在实现体中的连接,不应对原子构件本身对应的实现体作任何改动,最终生成的构件是一个带有实现体的多对象原子构件.

从第4.1节"原子构件的制作"可以看出,单对象构件只是多对象构件的一种特殊情况,其中的控制对象代表了原子构件负责对外的交互,并负责其他协作对象的实例化以及建立它们之间的关系.在对原子构件进行组装时,应生成一个新的C++的类,即针对复合构件所对应的控制对象,所有原子构件对应的控制对象作为协作对象,实现为这个新类中的嵌入对象.这个新类的生成完全符合第4.1.2节多对象构件中控制类对象的制作方法,其生成过程如下:

(1) 建立一个C++的类,类名对应复合构件名;

(2) 复合构件对外提供的功能对应到类的public部分中的函数声明;

(3) 构件对外要求的功能对应到类的protected部分的一个名为RequiredFunctions的函数数组;

(4) 在控制对象类的protected部分声明构件中包含的所有其他协作对象.为方便起见,这些协作对象是嵌入对象;

(5) 在控制对象类的public部分包括一个Initialize函数,不仅对其自身的RequiredFunctions数组进行初始化,而且调用各协作对象的Initialize函数,以此建立控制对象和协作对象各自的外部环境,即对外要求的功能.

(6) 实现对外提供的功能对应的各个函数,直接调用成员构件的映射函数即可,参数和返回值完全相同.

5 结论和今后的工作

在软件体系结构层次上对构件和构件之间的关系进行描述,可以较好地支持构件在接口处的集成,避免了构件的实现细节,从而增强了构件的可复用性,并通过对构件开发和系统集成的分工,为提高软件开发效率和质量提供了可能.

目前我们已经设计出青鸟构件描述语言,制定了可复用构件制作指南,并开发了可视化的系统建模和构件组装工具,支持对C++语言实现的构件进行组装.

本文的工作只是对基于软件体系结构的构件集成的初步研究,今后的研究工作包括:(1) 研究构件接口处定义的服务如何体现在原子构件的制作中,以及构件之间通过服务进行连接的语义;(2) 在进行系统建模和复合构件制作时,支持不同的体系结构风格;(3) 针对不同的目标环境和构件实现形态组装生成系统;(4) 考虑在异构平台上开发的构件之间的集成组装.

References:

[1] Tracz, W. Confessions of a Used Program Salesman-Institutionalizing Software Reuse. New York: Addison-Wesley Publishing Company, 1995.

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

[3] Yang, Fu-qing. Software reuse and relevant technology. Computer Science, 1999,26(5):1~4 (in Chinese).

[4] Luckham, D., Vera, J., Meldal, S. Three concepts of system architecture. Technical Report, CSL-TR-95-674, Stanford University, 1995.

[5] Bass, L., Clements, P., Kazman, R. Software Architecture in Practice. Reading, MA: Addison-Wesley Publishing Company, 1997.

[6] Zhang, Shi-kun. A study on hierarchical message bus based software architecture [Ph.D. Thesis]. Beijing: Department of Computer Science and Technology, Beijing University, 2000 (in Chinese).

附中文参考文献:

[3] 杨芙清.软件复用及相关技术.计算机科学,1999,26(5):1~4.

[6] 张世琨.基于层次消息总线的软件构架研究[博士学位论文].北京:北京大学计算机科学技术系,2000.

Building and Assembling Reusable Components Based on Software Architecture*

ZHANG Shi-kun, ZHANG Wen-juan, CHANG Xin, WANG Li-fu, YANG Fu-qing

(*Department of Computer Science and Technology, Beijing University, Beijing* 100871, *China*)

E-mail: zsk@cs.pku.edu.cn

http://www.cs.pku.edu.cn

Abstract: Software reuse and component-based software development are the inevitable trend in software engineering development and industrial production. Two important activities involved are the building and integrating of reusable software components. In this paper, the drawbacks of existing methods of component's integration are analyzed first, and a new method is proposed based on software architecture. Then, several different kinds of architecture and their influences to component integration are disscused. After that, a component model supporting interface connection architecture and plug and socket architecture, and an architecture-based component-assembling framework are proposed, and the building and assembling process of reusable software components is specified. Finally, the conclusion and further research direction are given.

Key words: software component; software architecture; component assembling

* Received February 5, 2001; accepted July 5, 2001

Supported by the Key Sci-Tech Project of the National 'Ninth Five-Year-Plan' of China under Grant No. 98-780-01