构件类的语法描述及构件关系的语义描述

顾 明
(深圳职业技术学院计算机系 深圳518055)

Syntax Description of Component Class and Semantics Description of Component Relationship

GU Ming
(Dept. of Computer, Shenzhen Profession Technology College, Shenzhen 518055)
gum@oa. szpt. net

Abstract In this paper, sameness and difference of component class and component, class and object are compared. Syntax of component class is described using BNF paradigm. After some definitions are given, semantics of component inherit and reference relationship are described.

Keywords Component class, Component, Syntax, Samantics

1. 概述

尽管O-O研究已经取得了不少成绩,但在应用软件开发中,真正使用O-O技术进行实际开发仍然存在不少的问题,其中的主要原因之一是:O-O对软件重用的支持主要通过继承机制,对用非O-O方法开发的软件,很难用O-O中的继承机制实现重用,因此,O-O对应用软件中存在的大量已有的软件无法重用;另一点是人们观念的转换需要一个逐步的过程。

为了克服O-O的不足,国际上已开始了构件软件的研究,希望软件像硬件一样能通过标准的组件来组装,从系统级和应用级上开始研究构件软件的规范化标准,目前是DCOM/COM+、JavaBean/Enterprise JavaBean和CORBA构件模型三足鼎立,构成了实现级构件模型工业标准的竞争与互操作并存的格局,实际上这三种工业标准,它们各自有其适用场合,谁也不能相互替代。

为了研究构件技术,在学术界和产业界出现了多种构件模型,3C模型[1,2]是一个得到普遍认可的模型,由概念(concept)、内容(content)、语境(context)三部分组成,该模型具有一定的宏观指导意义,但对构件的语法和语义没有给出具体的描述。

本文先从概念上说明构件类和构件类实例(构件)与O-O中的类和对象的异同,再结合我们开发应用软件的实际经验,给出应用软件中构件的BNF范式描述和构件关系的语义描述。

2. 构件类和构件的概念

构件和构件类概念的提出,第一是考虑到应用软件开发的实际需要,第二是受到面向对象中类和类实例概念的启发。

从直观上讲,构件相似于建筑上的预制板,而构件类相似于生产预制板的模子,通过模子来生产预制板,通过预制板来组合成高楼大厦。在应用软件的开发中,用构件类产生构件,通过构件的组装和控制来构造应用软件。

从计算机软件上讲,构件类是模板,它本身是一个静态的概念,但它可以生成各种动态的构件。在面向对象的程序设计中,构件类相当于类机制,构件就是由类生成的类实例——对象,在传统的程序设计中,构件相当于过程和其所使用数据的封装体。

构件的粒度可大可小,大的可以是工具、文件、目录和命名管道等,小的可以只有一个属性。

类和类实例都是程序设计语言中的概念,但它为我们提供了一个设计思想,在应用软件的开发中,我们基于类和类实例的思想,并对它们进行了扩充,把它们引入到设计阶段,来构造逻辑系统模型,这就是我们提出构件类和构件概念的原因。下面将说明对类和类实例是如何扩充的,即什么是构件类和构件。

这种扩充是从几个方面考虑的,首先,构件类和构件的概念不再仅仅限于编程阶段,作为一种设计思想,我们把它应用于设计阶段;其次,构件类和构件概念中可以没有继承机制,只要有封装即可;第三,通过聚集(Aggregation)和组装来支持重用;第四,通过构件操作界面(Interface)指针来实现构件之间的相互操作性;第五,构件类和构件之间的关系,不仅仅是类型和变量说明式的关系或调用创建函数和创建消息的关系,构件类可以通过一个生成器产生构件,而这种生成的方式可以根据实际应用的需要分为几种情况:

1)*变量类型和变量说明* 构件类相似于变量类型,构件相似于变量,生成器的作用是从变量类型到变量之间的转换,例如,C++中类和类实例的关系就是这种情况。

2)*函数生成* 调用一个特定的生成函数,由构件类生成构件,例如,Smalltalk中的New函数,Eiffel中的Create函数,这时的生成器相当于函数调用的处理过程。

3)*宏替换和预处理* 在构件类中可以有宏定义和预处理定义,对这些定义作宏替换或宏展开和预处理之后,就可以生成构件,这时的生成器就是一个作宏替换、宏展开和预处理的机制。在建立逻辑系统模型时,有些需要以后再加细说明的部分可以定义成宏替换和预处理。

4)*文件描述* 构件类也可以由一个文件的内容来描述,这时的生成器就是一个生成工具,通过生成工具,把由文件描述的构件类转换成构件。例如,在Informix-4GL中,可以通过屏幕表格说明文件来描述屏幕表格,然后,用屏幕表格生成工具生成最终的屏幕表格。

5)*规则表达式* 构件类中含有规则的描述,通过相应的

生成工具生成构件。例如，UNIX 中的 Lex 和 Yacc 就是由规则表达式生成程序的二个生成工具。

下面我们把构件类和构件类实例(构件)与 O-O 中的类和对象有什么异同做个比较：

1)在纯面向对象的设计中，类、封装和继承都是必不可少的，但对构件类而言，可以没有继承性，只要实现了封装性即可，也就是说，构件类放宽了对继承性的要求。从这一点上讲，构件类和构件不是 O-O 中的类和实例，而同基于对象(Object-Based)中的类和实例相似；

2)从对象和构件的生成方式上，在 O-O 中，类和对象之间的实例化过程比较单一，而构件类和构件之间的实例化过程包含了以上所述的五种情况，这对应用软件的开发是很有实用价值的。

3)从概念上讲，OOP 中的类和对象是编程时的概念，它们依赖于某种编程语言，而构件类和构件是设计时的概念，与具体的编程语言无关，同一个构件类，可由不同的编程语言实现；

4)从对软件重用的支持上，O-O 中的对象对软件重用的支持是通过继承机制体现出来的，而构件对软件重用的支持可以通过两个方面来体现，第一，在构件概念中，虽然主要强调封装，可以没有继承，但并不排除继承，在下面的构件类定义语言 CCDL 中，对继承留有接口；第二，构件对软件重用的支持主要是通过聚集(Aggregation)体现出来的[3]，本文也把聚集称为组装控制，它的含义是开发可重用的构件库，定义构件之间的相互操作标准(Interoperation Standards)，通过过程控制来组装生成应用程序；

5)从对象间和构件间的相互操作上，OOP 中对象间的相互操作通过公用接口来存取对象定义时的公用部分，这个公用部分可以是数据，也可以是操作函数，具体操作时，不同的 OOP 语言又有具体的规定；对构件而言，相互操作是通过构件操作界面(Interface)指针进行的，不允许直接操作构件中的数据，只能通过指针操作指定的界面，即在构件中，数据真正被封装了；

6)构件类和构件类实例(构件)是不同的，与 O-O 中的类和对象的不同类似，构件类为一组可能的构件的单纯静态描述，它是一个静态的概念，且只在静态描述中出现；构件为系统执行过程中创建的运行时的元素，是一个动态的概念，它在系统运行中出现，并占有一定的计算机存储单元。在描述中，我们只看到构件类，在运行时，我们只有构件。

3. 构件类的 BNF 范式描述

抽象构件类语法如下：

〈Comp_def〉 ::= Comp〈Comp_spec〉[〈Comp_body〉] END 〈Comp〉

由以上可见，一个构件类由构件类说明和构件类体二部分组成，其中，构件类体可以为空，把构件类说明和构件类体分开的原因是考虑到同一个构件类说明可以由不同的程序设计语言来实现，即同一个构件类可以有多个不同语言实现的构件类体。

〈Comp_spec〉::=Spec 〈Comp_name〉 构件类名
[〈Comp_para_lists〉] 参数表(可以为空)
〈Base_fields〉 构件类所属构件库名
〈Operation_lists〉 操作界面表
〈Relation_lists〉 关系表
〈Attribute_lists〉 属性表
END 〈Comp_spec〉

由以上可见，构件类说明以关键字 Spec 开始，以关键字 END 结束，最多可以包含六个部分，下面将逐一对这六个部分加以介绍。

3.1 构件类名的语法

〈Comp_name〉::=〈Identifier〉
〈Identifier〉 ::= Letter Letter| Digital| '-' *

构件类名由标识符标识，标识符是以字母打头的字母数字或下划线串。

3.2 参数表的语法

〈Comp_para_lists〉::= Para(〈Comp_para〉,〈Comp_para〉*)
〈Comp_para〉::=〈Identifier〉

参数表由多个参数组成，若参数表非空，则它至少包含一个参数，而参数可由标识符标识。一个构件类的参数表可以为空，构件类带参数称为含参构件类，用实参替换形参，即可使含参构件类成为一般构件类。

以下凡是"[ ]"中的内容表示可以为空。符号"*"是 BNF 中的含义。

3.3 构件类所属构件库名的语法

〈Base_fields〉 ::= Base〈Base_lists〉
〈Base_lists〉::= 〈Base_name〉,〈Base_name〉* 〈Base_name〉 ::=〈Identifier〉

构件类库名提供了一种将相关的构件类按照某种特征组织在一起的手段，这种特征可以是应用领域等，一个构件类也可属于多个构件类库，这样虽然带来信息冗余，但有时却可为实际应用提供方便。

3.4 操作界面的语法

〈Operation_lists〉 ::= Operation〈Operation_declaration〉,*
〈Operation_declaration〉 ::= 〈Operation_name〉〈Operation_para〉
[:〈Return_values〉]
〈Operation_name〉 ::= 〈Identifier〉
〈Operation_para〉 ::= 〈Comp_para_lists〉
〈Return_values〉 ::= 〈Identifier〉

以上说明，操作由操作名、操作参数和操作返回值三部分所组成，其中，操作参数就是该构件类参数表中的参数，但某个具体的操作可能只用了参数表参数的一部分，一个操作可以没有操作参数，即参数表为空。操作界面是构件类对外部的唯一接口。

3.5 关系表的语法

〈Relation_lists〉 ::= Rela[〈Inherit_R〉] 〈Reference_R〉

构件类之间的关系有两种：继承关系和引用关系，其中继承关系可以为空。

3.5.1 继承关系

〈Inherit_R〉 ::= Inherit〈Supercomp_lists〉 (1)
〈Supercomp_lists〉 ::= 〈Supercomp〉; * (2)
〈Supercomp〉 ::=〈Comp_name〉[〈Rename_clause〉][〈Redefine_clause〉]
〈Rename_clause〉 ::= Rename 〈Rename_part〉,〈Rename_part〉* (3)
〈Rename_part〉 ::= 〈Old_name〉 As 〈New_name〉
〈Old_name〉 ::= 〈Attribute_name〉 | 〈Operation_name〉
〈New_name〉 ::= 〈Identifier〉
〈Redefine_clause〉 ::= Redefine 〈Redefine_name〉〈Redefine_name〉 * (4)
〈Redefine_name〉 ::= 〈Operation_name〉

在上面继承关系的语法中，继承的父构件类是以 Inherit 开始说明的，由式(4)可以看出，允许多重继承，因而一个构件

类可以有多个父构件类，子构件类继承所有父构件类的全部静态和动态特征。

由于允许子构件类有多个父构件类，因而名字冲突将是不可避免的，为此，提供了由关键字引导的换名子句，如式(3)所示它将父构件类中的属性或名字按需要改成新名字。

有时子构件类还需要细化或增加父构件类的操作界面以满足自己的需要，为此式(4)提供了重定义 Redefine 子句指示将父构件类的某个操作界面重定义。

继承关系可以为空，即可以没有继承关系，在语法中给予描述，是为以后构件类的进一步研究留下扩充的接口。

3.5.2 引用关系

```
〈Reference_R〉 ::= Refe〈Comp_name〉〈Interface〉
〈Comp_name〉 ::= 〈Identifier〉,*
〈Interface〉::= 〈Operation_name〉,*
```

上式说明，一个构件类可以引用另一个构件类的操作界面，操作界面是操作界面语法描述中的操作名。当用户引用操作界面时，先使用 Queryinterface 界面查询构件类界面的情况，然后再决定要使用哪个界面。

3.6 属性的语法

```
〈Attribute_lists〉 ::= Attr〈Attr_def〉;*
〈Attr_def〉 ::= In〈Inside_attr〉 Env〈Env_clause〉
〈Inside_attr〉 ::=〈Primitive_attr〉;* [〈Create_time〉][〈Storage_
                priority〉]
                [〈Modify_time〉][〈Use_frequency〉]
〈Primitive_attr〉 ::= Interage | Real | Character |
                Boolean | String | Array | record
〈Env_clause〉 ::=〈Version_spec〉〈Develop_env〉〈Run_env〉
                〈Readme_spec〉〈Documetation_name〉
〈Version_spec〉 ::= Version"〈Version_description〉"
〈Develop_env〉 ::=Develop〈Source〉;〈Library〉;〈Compiler〉;
                〈Database〉;〈Netwark〉;〈Operating system〉
                〈Source〉 ::= 〈Language_name〉
〈Library〉 ::= 〈Library_name〉,〈Library_name〉*
〈Compiler〉 ::= 〈Compiler_name〉
〈Database〉 ::= 〈Database_name〉
〈Network〉 ::= 〈Network_name〉
〈Operating system〉 ::= 〈Operating system_name〉
〈Run_env〉 ::= Run〈Operating systm〉;〈Database〉;〈Network〉
〈Readme_spec〉 ::= Readme"〈Readme_description〉"
〈Documention_name〉 ::= 〈Identifier〉,*
```

上面说明，属性由内部属性和环境属性二部分组成，其中，内部属性包括预先定义的一些原始类，如整数、实数、字符、布尔型、字符串、数组和记录，还包括创建时间、存取权限、修改日期和使用频度。

环境属性包括版本说明，构件类开发环境，构件运行环境，构件类自然语言理解和与该构件类相关的文档名称。

3.7 构件类体说明的语法

```
〈Comp_body〉 ::= Body〈Comp_obj〉〈Comp_demo〉 END
                〈Comp_body〉
〈Comp_obj〉 ::= Obj[〈Comp_para_lists〉][〈Local_para_lists〉]
                〈Body_implementation〉
〈Comp_demon〉 ::=Demon 〈Comp_ins_name〉
                〈System_configuration〉
〈Comp_ins_name〉 ::= 〈Identifier〉
〈System_configuration〉 ::= Sys〈Invoke_comp_name〉
                〈Pinvoke_comp_name〉
〈Invoke_comp_name〉 ::= Invoke〈Identifier〉,*
〈Invokep_comp_name〉 ::= Invokep〈Identifier〉,*
```

以上说明，构件类体由两部分组成，一部分是构件类说明中操作界面的具体实现例程，它由参数表中提供的操作参数和例程具体实现时的局部参数为数据结构来实现每个具体的操作，这两种参数可以同时为空，也可以某一个为空。每个操作的具体实现例程，可以用某种计算机编程语言来具体实现。同一个构件类的操作界面可由不同的编程语言来实现，如 C++，4GL 等；另一部分为构件类的演示，由于构件类本身是不可执行的，所以要演示某个构件类的功能，必须首先使之实例化，再加上必须的系统配置后方可实际运行，系统配置包括引用构件和被引用构件等。

4. 构件关系的语义

4.1 构件关系语义的有关概念

构件的关系语义是指构件之间的一种静态关系，从构件的语法描述可知，构件之间的关系有二种：继承关系和引用关系，为了描述这二种关系的语义，先引入以下几个定义：

定义1(事件) 是构件操作发生的触发源，是瞬间发生的动作，事件可以是函数调用，也可以是消息、指令等。

定义2(构件的状态) 指在某一给定时刻，构件的属性值和操作返回值二部分的组合，构件状态的改变是由于构件操作的执行，而操作是由事件来触发的。

定义3(构件的生命周期) 所有构件状态改变的集合，组成了该构件的生命周期，由于状态改变的基本原因是事件，因此，构件的生命周期可以被定义为触发所有构件状态改变的事件的集合。

对某个构件 C，它的生命周期被定义为：

$LC(c)=EVi(v)\quad i=0,1,\cdots,n$

其中，事件 EV0相应于构件 C 的创建，事件 EVn 相应于构件 C 的删除。

例如，构件 C 和 C' 分别为两个命令构件，则：

LC(c)＝命令出现，命令传达，命令执行

LC(c')＝命令出现，命令取消

事件可以按时间顺序来排序，EVi ＜ EVj 当且仅当 EVi 在 EVj 之前发生，在某一给定时刻，仅有一个事件可以发生，所以 EVi ＝EVj 当且仅当 EVi 和 EVj 是相同的事件。

由此可得到构件生命周期的蕴含关系如下：

$LC(c')\cup LC(c) \Leftrightarrow EV0(c') >= EV0(c)$ and $EVn(c') <= EVn(c)$

上式说明，若构件 C' 的生命周期蕴含于 C 的生命周期之中，当且仅当，构件 C' 的创建比 C 要迟，而 C' 的删除比 C 要早，即 C' 的生命周期比 C 要短。

定义4(构件类模式) 是构件类的语法描述中构件类说明的六个部分组成的六元组：

Sch(A) ＝ (Name,Para,Base,Rela,Attr,Operation)

上式说明，构件类 A 的模式由构件类名称，构件类参数，构件类库名，构件类关系，构件类属性和构件类操作界面六个部分组成。

定义5(构件类实例集) 是指由构件类所生成的所有实例的集合，若某构件类的构件类模式是 Sch(A)，那么相应的构件类实例集为 Ins(A)。

4.2 构件关系的语义描述

4.2.1 继承的语义 构件之间按照继承关系组成一个继承的层次结构，构件之间的继承关系是“一般”和“特殊”的关系。若有一个继承关系，它的特殊构件类模式 Sch(S)的所有实例继承一般构件类模式 Sch(S)的所有实例，当且仅当下列三个公理成立。

(1)模式蕴含公理：

Sch(G) Sch(S)

这个公理说明构件类模式 Sch(S)继承 Sch(G)的构件类名称，构件类参数，构件类库名，构件类关系，构件类属性和构件类操作，但 Sch(S)有自己不同于 Sch(G)或增加的部分，即 Sch(G)有的 Sch(S)都有，但 Sch(S)有的 Sch(G)不一定都有。如果 Sch(S)有的 Sch(G)也都有，则 Sch(G)＝ Sch(S)，那

么 Sch(S)也就没有存在的必要了。

(2)实例蕴含公理

Ins(G) Ins(S)

上式说明,每个时刻,对于特殊构件类模式产生的每个实例,都有相应的一般构件类模式实例存在,一个一般构件可被多个特殊构件所继承,即一个父亲可以有多个儿子。

(3)生命周期蕴含公理

LC(S) LC(G)

上式说明,特殊构件的生命周期比一般构件的生命周期要短,即特殊构件的生命周期蕴含于一般构件的生命周期之中。

4.2.2 引用的语义 引用是构件类和构件之间一种临时性的关联,如果构件类 A 引用了构件类 B,或者说 B 被 A 引用,称 A 为引用构件类,B 为被引用构件类,那么,当且仅当以下叙述成立:

假设存在一个函数 Ref,它返回构件类 B 的实例 b 的集合,这些实例集合在 A 的生命周期中被 A 的实例 a 引用,

则: $b\in Ins(B) / a\in Ins(A)$,有 $b\in Ref(a)$:

$EV0(a)\neq EV(b)$ or $EVn(a)\neq EVn(b)$

$a\in A, b\in Ref(a) / LC(a)\cup LC(b)$

上面说明的直观含义是,引用构件的生命周期蕴含在被引用构件的生命周期之中,即引用构件的生命周期比被引用构件的生命周期短。

总结 构件的 BNF 范式描述了构件的语法,构件的关系语义描述了构件之间关系的含义,本文通过对构件的语法和构件的关系之间的语义的描述,使对应用软件中的构件的概念有一个比较全面的认识,这对于用构件组装应用软件有一个概念上的指导。

参 考 文 献

1 Edwards S H. Toward a model of reusable software subsystems. In: Philbrick S, Stevens M, eds. Proc. of the 15 th workshop on Software Reuse. Larry Latour, 1992

2 Tracz W. Implementation working group summary. In: Baldo J, ed. Reuse in Practice Workshop Summary. Alexandria. 1990. 10~19

3 Brockschmidt K. Inside OLE2. Microsoft Press, 1994

4 Brown A. Large-Scale Component-Based Development. New Jersey: Prentice Hall, Inc., 2000

5 Blom M, Nordby E J. Semantic integtity in component based development. Project Report, Sweden: Malardalen University, 2000

6 Object Management Group(OMG). The Common Object Request Broker: Architecture and Specification V2.3. 1999. http://www.omg.org

7 贾育,顾毓清. 基于领域特征空间的构件语义表示方法. 软件学报, 2002,13(2):311~316

8 杨芙清,梅宏,李克勤. 软件复用与软件构件技术. 电子学报,1999, 27(2):69~75

(上接第146页)

表1 数字水印对常见图像处理与攻击的抵抗能力

攻击方式	参数	归一化相关系数	攻击方式	参数	归一化相关系数
叠加噪声	随机分布(10%)	0.991	几何剪切	二分之一	0.817
	均匀分布(5%)	0.993		四分之一	0.892
JPEG 压缩	质量系数为70%	0.907	图像增强	锐化	1.000
	质量系数为80%	0.852		边缘锐化	1.000
平滑滤波	低通滤波	0.954	马赛克效果	2×2块	0.897
	中值滤波	0.891		3×3块	0.702

子块、自适应修改纹理子块像素灰度值等措施,将随机置换的二值标志图像水印信息自适应地嵌入到原始图像的像素值中,从而有效解决空域数字水印技术普遍存在的透明性较差、抗攻击能力较弱等问题。实验结果表明:该数字水印算法不仅具有较好的透明性,而且对诸如叠加噪声、JPEG 压缩、平滑滤波、几何剪切、图像增强、马赛克等攻击均具有较好的鲁棒性;同时,具有算法简洁高效、易于实时处理等优点。

参 考 文 献

1 Lu C S, Liao H Y M. Multipurpose watermarking for image authentication and protection [J]. IEEE Trans. on image processing, 2001, 10(10): 1579~1592

2 Cox I J, Miller M L. The first 50 years of electronic watermarking [J]. Journal of Applied Signal Processing, 2002, (2): 126~132

3 易开祥,石教英,孙鑫. 数字水印技术研究进展[J]. 中国图像图形学报, 2001, 6(2): 111~117

4 Hartung F, Girod B. Watermarking of MPEG-2 encoded video without decoding and re-encoding [C]. In: Freeman M., Jardetzky P., Vin H. M, els. Proc. of the SPIE conf. on Multimedia Computing and Networking, Vol 3020. Bellingham: SPIE Press, 1997. 264~273

5 Cheng Yen Hui. Watermark embedded in permuted domain [J]. Electronics Letter, 2001, 37(2): 80~81

6 Kankanhalli M S, Rajmohan, Ramakrishnan K R. Content based watermarking of image [C]. In: Effelsberg W.. ACM Multimedia Electronic Proc. the 6th ACM Intl. Multimedia Conf. New York: ACM Press, 1998

7 Chiou-Ting H, Ja-Ling W. Hidden digital watermarks in images [J]. IEEE Trans. on Image Processing, 1999, 8(1): 58~68