接口连接式构件组装的一种形式化方法*)

孙 莹 陈松乔
(中南大学信息科学与工程学院 长沙 410083)

摘 要 构件组装是基于构件的软件开发的研究重点之一,能够有效地提高软件开发的效率和质量。以往大部分构件组装技术是在"成功组装路线"的前提条件下实现的,缺乏对构件组装正确性的检验。本文改进了常用的接口连接式构件组装技术,采用形式化方法描述和推导与构件以及构件组装相关的问题,给出了映射算法,实现了从构件组装规约向粘合代码的自动转换,为构件组装形式化分析、组装正确性检验提供了保证。

关键词 基于构件的软件开发,构件组装,接口连接式组装,组装推导

A Formal Method of Interface-Connection Component Composition

SUN Ying CHEN Song-Qiao
(College of Information Science and Engineering, Center South University, Changsha 410083)

Abstract Component composition is one of the emphasis of research in component based software development. It can greatly improve the efficiency and quality in software development. Currently, most of the component composition technology is implemented under the condition of "successful traces", ignoring the check on the correctness of composition. This paper optimizes the component interface-connection technique, and adopts formal methods to describe and deduce the problems about components and their composition. Based on this work, a mapping algorithm is brought out to implement the automatic conversion from composition specification to glue code. The research makes preparations for formal analysis on component composition and the validation of the correctness of composition.

Keywords Component-based software development, Component composition, Interface connection composition, Composition deduce

1 引言

近年来,基于构件的软件开发(CBSD—Component Based Software Development)技术越来越成熟和规范化,它涉及到构件的获取、描述、分类、存储、检索、组装等几个方面的问题。而软件构件组装正是这些问题中的重点。正如文[1]中所言:"在基于构件的软件开发中,系统开发的重点已从程序设计变成了构件组装"。构件组装的基本思想是通过"粘合代码"或"连接件"将已有的构件组合在一起,以提供粒度更大和功能更齐全的功能构件,并有效地支持软件复用。目前,国内外针对构件组装技术的研究主要集中在体系结构领域,如北大贝尔试验室提出的基于体系结构描述语言 ABC/ADL 的构件组装[2],基于 Darwin[3]、Wright[4]、Unicon[5]等体系结构描述语言的构件组装等等。纵观现有的构件组装技术以及辅助工具产品,可以发现:它们大都采用了接口连接式构件组装技术,即通过构件接口来完成系统中构件之间的连接组装[6]。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,因此通过接口就可以定义系统中构件之间的所有连接。而且,构件之间的连接只有在所要求的功能和其它构件所提供的功能一致的情况下才能够建立。通过这种组装方式,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。如图 1 所示,一般的 C/S 信息管理系统可初步分解为"应用界面"、"数据库访问"和"日志管理"3个构件,其中应用界面构件负责产生数据库操作内容,数据访问构件负责对数据库进行操作,同时通过日志管理构件登记所有的数据库操作信息。3 个构件之间通过构件接口完成相互之间的连接组装。

图 1 构件连线组装示例

但目前的接口连接式构件组装是在"成功组装路线"的前提条件下实现的[7],忽略了构件组装过程中可能出现的重复连接、异常连接、闭环连接等潜在错误,缺乏对构件组装过程的正确性验证。在本文中,主要针对接口连接式构件组装机制进行了改进,采用了形式化语义的方法描述和推导了与构件以及构件组装相关的问题,并在此基础上给出了映射算法,以便实现构件组装规约向粘合代码的自动转换。采用形式化方法对构件组装相关问题进行描述有利于构件组装推导,为构件组装形式化分析、组装正确性的检验提供了保证。

2 构件的形式化定义

构件是 CBSD 的基本元素。我们需要将构件组装所关心

*)本课题得到高校博士点专项科研基金([2003]172)资助。孙 莹 博士生,讲师,主要研究领域为软件复用技术、构件组装技术;陈松乔 教授、博士生导师,主要研究领域为软件工程。

的构件结构、构件形态和表示方法加以标准化，才能使关心和使用构件的外部环境（如使用构件构造出的应用系统、构件组装辅助工具和构件复用者等）能够在一致的概念模型下观察和使用构件[2]。针对不同的需求，构件的标准化方法也各不相同。在本文中，我们将构件标准化为静态结构与动态结构两个部分。为了有利于后面的构件以及构件组装形式化的探讨，我们将图 1 中的数据库访问构件 CDBAcess 进一步细化，主要由数据库操作构件 CSQLOperation、数据库连接池构件 CpoolManager 连接组装而成，如图 2 所示。

(1)构件的静态结构。即构件框架结构。它由一系列接口构成，构件接口是构件与外界进行信息通讯的唯一通道，每个接口都对应一个具体的操作序列，表示该接口所能提供的操作方法。构件接口按照提供服务的方式不同，可以分为两种类型：一是向外提供服务的 S 型接口，如图 2 中构件 CSQLOperation 所提供的 IDBAcc，构件 CPoolManager 所提供的 IDBConnection 接口；另一种是需要外界对构件提供服务的 R 类型接口，如图 2 中构件 CSQLOperation 所需的 IDBLog、IDBConnection2 个接口。只有环境满足了构件的 R 类型接口，才能使构件对外提供 S 类型的接口服务。在一个标准的构件中，S 类型的接口是必须有的，表示构件能对外提供特定的服务，而 R 类型的接口不是必须存在的，表示某些构件可以不需要外界的支持就能独立地提供服务。

(2)构件的动态结构。即构件行为描述，对外表现为所能提供的服务功能集，主要是通过构件接口中的操作序列来表现。

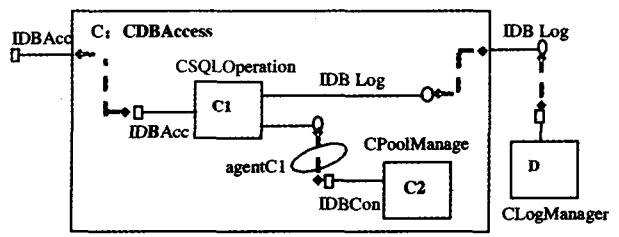

图 2　数据库访问构件

通过以上分析，具体构件的形式化定义如下：

定义 1(构件框架结构)　我们定义构件的框架结构为一个元组表达式 $F=(S,R,OP)$，其中：

- $S\subseteq I$，构件对外界提供的服务接口集
- $R\subseteq I$，构件所需外界提供的服务接口集
- 该构件的接口标识集 $I=S\cup R$，并且 $S\cap E=\phi$
- OP 是构件接口对应的操作序列集合，并且 $i: I\rightarrow op(i)$。$op(i)$ 表示与接口 i 对应的操作序列。

定义 2(构件功能规约)　我们定义 V 是构件所有功能 v 的集合，并且针对每个 $i\in I$，$v(i)\in op(i)^*$。所以 $V=\{v|\bigcup_{i\in I}v(i)\in op(i)^*\}$。

定义 3(构件)　我们定义构件为一个元组表达式 $C=(F,B)$，其中：

- F 为构件的框架结构
- B 为构件的行为描述，并且 $B\subseteq V, B\neq\phi$。

如图 2 所示，SQL 操作构件 CSQLOperation 与连接池管理构件 CPoolManager 组装可形成数据库访问构件 CDBAccess，为外界提供通用的数据库访问功能，分别用 $c1$、$c2$、c 来标识这 3 个构件。该组装涉及到的服务接口主要为数据库访问 IDBAcc、数据库日志 IDBLog、数据库连接 IDBCon。其中，IDBAcc 接口是构件 $c1$ 对外提供的服务接口，包括执行查询 query、执行更新 update、执行存储过程 storedprocedure、日志事件 logevent、日志查询 logsql5 个操作方法，在这里我们用 $a1$、$a2$、$a3$、$a4$、$a5$ 来分别标识这 5 个操作；IDBLog 接口是构件 $c1$ 的 R 型接口以及构件 D 的 S 型接口，包括获取日志句柄 getloghandle，释放日志句柄 releaseloghandle 两个操作方法，分别标识为 $b1$、$b2$；IDBCon 接口是 $c1$ 构件的 R 型接口以及 $c2$ 构件的 S 型接口，主要包括获取数据库连接 getconnection、释放数据库连接 freeconnection 两个操作，分别标识为 $c1$、$c2$。

在图 2 中，$c1$ 构件需要外界环境提供 IDBLog、IDBCon 这两个接口服务才能对外界提供 IDBAcc 接口服务。根据定义 1，$S_{c1}=\{IDBAcc\}$，$R_{c1}=\{IDBLog, IDBCon\}$，并且 $S_{c1}\cap R_{c1}=\phi$，$I_{c1}=S_{c1}\cup R_{c1}$，所以接口集构件 c 的框架结构为：$F_c=(S_c,R_c,op_c)$，其中 $S_c=\{IDBAcc\}$，$R_c=\{IDBLog\}$，$op_c(IDBAcc)=\{a1,a2,a3,a4,a5\}$，$op_c(IDBLog)=\{b1,b2\}$。行为集为：$B_c=\{(a1a2a3,\wedge),(a1a4a5,b1b2),(a2a4a5,b1b2),(a3a4a5,b1b2),(a1a2a3a4a5,b1b2)\}$。

由此可见，复合构件 c 的组装达式可简化为：$c=\|_{c_i\in AC}c_i$。根据参与组装构件的框架结构以及行为，按照以上定义可预先推导出复合构件的框架结构以及行为，并可用于系统行为分析接口兼容性的检测，达到检验构件组装方案的正确性、提高组装软件系统的质量的目的。

3　构件以及构件组装的形式化描述

3.1　构件形式化描述

图 3 给出了构件描述的 BNF 范式。interface-specification-section 部分罗列出构件的接口规约，在接口规约中定义了操作方法集；provides-interface-section 部分描述了构件对外提供的 S 类型的服务接口，requires-interface-section 部分描述了构件所需的 R 类型的服务接口。这 3 个部分构成了构件的框架结构。behaviors 服务描述了构件所能提供的功能集，即构件行为描述。

```
frame ::=
    Frame frame-name
        interface-specification-section
        provides-interface-section
        [requires-interface-section]
    End Frame
component ::=
    Component comp-name
        Frame-name
        [behaviors]
    End Component
```

图 3　构件描述的 BNF 范式

```
compositionSpec ::=
    compositionSpec compo_name
        instants-list-section
        bind-list-section
        subsume-list-section
        delegate-list-section
    End compositionSpec
```

图 4　构件组装规约描述 BNF 范式

3.2 构件组装的形式化描述

构件组装方案可由组装规约描述来定义，图 4 给出了组装规约描述的 BNF 范式。其中，instants-list-section 部分描述了构件实例的列表，bind-list-section 部分、subsume-list-section 部分、delegate-list-section 部分分别描述了构件接口之间的 bind、subsume、delegate3 种通信接口映射表。

```
compositionSpec C {
  inst :
      CSQLOperation c1;
      CPoolManage   c2;
  bind :
      c1.IDBConnection to c2.IDBConnection
      using agentC1;
  subsume :
      c2.IDBLog to C.IDBLog;
  delegate :
      c1.IDBAcc to C.IDBAcc;
};
```

图 5 构件 c 的组装规约描述

但在实际的构件组装中，构件直接进行装配非常困难，主要原因在于：(1)从市场上购买的商用构件或从网上下载的免费构件由不同的构件开发商提供，构件缺少一致的接口标准；(2)组装构件仅有接口的语法匹配是不够的，因为两个具有相同事件或属性名的构件，其功能实现可能差异很大。由于这些问题的存在，构件接口之间的映射连接需要进行调整。在这里，我们使用了一种基于 Agent 适配器，通过半自动化配置，完成构件接口之间的连接映射(如图 2 中的 agentC1)。为了简化构件组装方案的实现，我们将这种起到桥梁作用的 Agent 适配器同样看成具有 S 类型、R 类型接口的标准构件。构件 *c*1 与 *c*2 的组装由于中间加入的一个适配器，因而转化为 *c*1、*agentc*1、*c*2 这 3 个构件之间的组装，其中 *agentc*1 的 S 类型接口与构件 *c*1 的 R 接口进行映射，*agentc*1 的 R 接口与构件 *c*2 的 S 接口进行映射，然后在 *agentc*1 内部，通过相关的配置完成自身的 R 与 S 接口的映射转换过程。在组装规约描述中，通过使用关键字 using 表示用于接口转换的 Agent 适配器。图 5 中给出了图 2 中复合构件 *c* 的组装规约描述实例。

```
void ComSpecTemplate( compositionSpec )
{
  //对组装规约中的语句，如 c1.IDBConnection to c2.IDBConnection using agentC1 有如下定义：
  // c1 为目标对象 DestinationObject，c2 为源对象 SourceObject
  // c1.IDBConnection 为目标对象接口 DestinationObject.Interface; c2.IDBConnection 为源对象接口 SourceObject.Interface
  // agentC1 为代理 Agent

  //1.定义变量
   define $newClassName,$inst[],$subsume[],$bind[],$delegate[];

  //2.析解组装规约 compositionSpec，将规约中的 4 个 section 的数据分别保存到 inst，bind，delegate 和 subsume 四个数组中
   ExtractSpec( compositionSpec,$newClassName,$inst,$subsume,$bind,$delegate);

  //3.对 delegate 中的每一项做如下处理：分离出目标对象的接口，并调用方法 CreateInterface 产生该接口的代码。
    for all $a in $delegate do
      CreateInterface(a.DestinationObject.Interface);

  //4.构造组装规约指定的新的复合构件类$newClassName，并实现所有的 delegate 中的目标对象的接口
    CreateClass($newClassName, $delegate[].DestinationObject.Interface );

  //5.扫描数组 inst，在新的复合构件类的构造函数中建立 inst 数组中指定的所有对象实例。
    for all $a in $inst do
     AddAttribute( $newClassName, a.ClassType, a.InstanceName);

  //6.扫描数组 bind，该数组描述了参与组装的构件之间的连接关系，如果在连接之间定义了 using agent，
  // 则需要建立代理类进行两个构件之间的连接
    for all $a in $bind do
     if( $a.existAgent == true ){
       CreateUserDefineClass( $a.Agent, $a.DestinationObject.Interface);
       for( all $method in $a.DestinationObject.Interface )
         AddMethod( $a.Agent, $method)
       AddAttribute($newClassName, $a.Agent )
       AddAttributeConnection($a.Agent.Interface, $a.SourceObject);
       AddAttributeConnection($a.DestinationObject.Interface, $a.Agent);
     }
     else
       AddAttributeConnection($a.DestinationObject.Interface, $a.SourceObject);

  //7.扫描数组 subsume，对数组中的每一项进行分解出源对象和目标对象，直接在新的复合构件的构造函数中建立连接代码：
    for all $a in $subsume do
      AddAttributeConnection($a.DestinationObject, $a.SourceObject);

  //8.扫描数组 delegate，对 delegate 数组中的每一项进行如下处理：
  //  分解出源对象和目标对象，在新的复合构件类中，实现接口源对象接口中的方法
     for all $a in $delegate do {
       for all $method in $a.SourceObject.Interface
         AddMethod( $newClassName, $method ){return $a.DestinationObject.Interface.Method;};
     }
}
```

图 6 CJMA 实现过程

4 构件组装规约映射算法

通过以上阐述的构件、构件组装的形式化定义以及相关的形式化推导，可以验证并生成正确的构件组装规约。在此基础上，我们采用了组装规约映射算法 CJMA，来实现复合构件的组装规约向 Java 源码的映射转换。该算法通过析解组装规约脚本，自动生成 Java 粘合代码，将参与组装的协作构件粘合在一起，形成新的复合构件源码。源码编译通过之后，

可以将编译生成的复合构件实体添加至构件库中，以便今后的构件复用。该算法的执行可以在复合构件的组装约生成之后或在整个应用(系统)运行之前。具体的执行步骤如图 6 所示。

相关工作及结论 构件组装是构件化软件开发的重点，软件的需求分解、体系结构、构件模型、构件粒度、构件评估标准、市场供应都对构件组装产生一定的影响，但其中最为核心的问题是如何采用有效的组装方法来支持构件化软件的开发。目前，构件的组装方法与技术有多种，根据组装的构件粒度以及应用环境的不同，组装技术主要集中在构件接口之间的静态连线组装、服务构件之间动态流程组装这两个方面。接口连接式构件组装是这两类组装的基本组装技术，而其中的构件组装形式化描述以及组装方案正确性的检验，将直接影响到构件组装的效果。本文主要针对构件接口连接式组装技术，采用形式化方法描述与推导了构件组装的相关问题，为构件组装的形式化分析以及组装正确性的检验打下了基础。用户可以根据应用需求，遵循体系结构的思想，自顶向下定义各层复合构件的组装规约。通过形式化分析并验证通过之后，采用构件组装规约映射算法 CJMA，自动生成源代码并编译产生新的复合构件，自底向上地快速搭建应用系统。通过这种技术可以规范基于构件的软件开发，有利于提高系统的开发效率和质量。

参 考 文 献

1 Clements P C. From Subroutines to Subsystems: Component-Based Software Development. In: Component-Based Software Engineering: Selected Papers from the Software Engineering Institute. IEEE Computer Society Press, 1996. 3～6

2 梅宏，陈锋，等. ABC：基于软件体系结构、面向构件的软件开发方法. 软件学报，2003，14(4)：721～732

3 Department of Computing. The Darwin Language. 3d edition. Imperial College of Science, Technology and Medicine, September 1997

4 Allen R, Garlan D. A formal Basis for Architectural Connection. ACM Transactions on Software Engineering and Methodology, 1997, 6(3): 213～249

5 Shaw M, DeLine R, Klein D, et al. Abstractions for Software Architecture and Tools to Support Them. IEEE Transactions on Software Engineering, 1995, 21(4): 314～335

6 张世琨，张文娟，常欣，等. 基于软件体系结构的可复用构件制作和组装. 软件学报，2000，12(9)：1351～1359

7 Adamek J, Plasil F. Component Composition Errors and Update Atomicity: Static Analysis. Accepted for publication in the Journal of Software Maintenance and Evolution: Research and Practice, 2004

8 Visnovsky S. Modeling Software Components Using Behavior Protocols: [Ph D. Thesis]. advisor: Frantisek Plasil, Dec. 2002

9 Tansalarak N, Claypool K T. XCompose: An XML-Based Component Composition Framework. Third International Workshop on Composition Languages, 2003

10 任洪敏，钱乐秋. 构件组件及其开式化推导研究. 软件学报，2003，14(6)：1066～1074

(上接第 232 页)

值，如果它们满足操作 Join 的前置条件，则在后置条件中用 currentstate(o)加到后状态变量前，把后状态的标志"′"取消，用 oldstate(o)加到前状态变量前，用"."分开，我们有：

$$\{\text{dtype. o} = \text{Queue}[T] \wedge \text{Join. pre}_{\text{Queue}[T]}(\text{currentstate}_{(o)\text{queue}[T]})\}$$
$$\text{o. Join}_{\text{Queue}[T]}()$$
$$\{\text{currentstate(o). items} = \text{oldstate(o). items} \frown \langle \text{item?} \rangle$$
$$\wedge \text{currentstate(o). size} = \#\text{currentstate(o). items}$$
$$\wedge 0 \leqslant \text{currentstate(o). size} \leqslant \text{max}\}$$

现在可以考虑推理的重用。如果现在知道了 o 所指的对象的类型是 CQueue [T]，且此对象的状态满足操作 Join 在类 CQueue [T]上的前置条件，我们有：

$$\{\text{dtype. o} = \text{Queue}[T]$$
$$\wedge \text{otype} = \text{CQueue}[T] \wedge \text{Join. pre}_{\text{Queue}[T]}(\text{currentstate(o)})$$
$$\wedge \text{Join. pre}_{\text{Queue}[T]\text{CQueue}[T]}(\text{currentstate(o)})\}$$
$$\text{o. Join}_{\text{CQueue}[T]}()$$
$$\{[\text{currentstate(o). items} = \text{oldstate(o). items} \frown \langle \text{item?} \rangle$$
$$\wedge \text{currentstate(o). size} = \#\text{currentstate(o). items}$$
$$\wedge 0 \leqslant \text{currentstate(o). size} \leqslant \text{max}$$
$$\wedge \text{currentstate(o). count} = \text{oldstate(o). count} + 1$$
$$\wedge 0 \leqslant \text{currentstate(o). count}]$$
$$\Leftrightarrow \text{Join. post}_{\text{Queue}[T]} \wedge \text{Join. post}_{\text{Queue}[T]\text{CQueue}[T]}\}$$

即只要验证对象目前状态满足 Join. $\text{pre}_{\text{Queue}[T]\text{CQueue}[T]}$ 即可，如果它满足，那么它的后状态就满足 Join. $\text{post}_{\text{Queue}[T]}$ 和 Join. $\text{post}_{\text{Queue}[T]\text{CQueue}[T]}$ 部分后置条件，把它们合取就可以得到后置条件 Join. $\text{post}_{\text{CQueue}[T]}$，这就体现推理重用。

(2)对于 o. Leave，它适合情况 3. 2. 3(1)，按上述操作，我们有：

$$\{\text{dtype. o} = \text{Queue}[T] \wedge \text{Leave. pre}_{\text{Queue}[T]}(\text{currentstate(o)}_{\text{Queue}[T]})\}$$
$$\text{o. Leave}_{\text{Queue}[T]}()$$
$$\{\text{oldstate(o). items} = \langle \text{item!} \rangle \frown \text{currentstate(o). items}$$
$$(\text{currentstate(o). size}$$
$$= \#\text{currentstate(o). items} \wedge 0 \leqslant \text{currentstate(o). size} \leqslant \text{max}\}$$

如果知道了 o 所指的对象的类型是 CQueue [T]，我们有：

$$\{[\text{dtype. o} = \text{Queue}[T] \wedge \text{otype} = \text{CQueue}[T]$$
$$\wedge \text{Leave. pre}_{\text{Queue}[T]}(\text{currentstate(o)})$$
$$\wedge \text{Leave. pre}_{\text{Queue}[T]\text{CQueue}[T]}(\text{currentstate(o)})]$$
$$\Leftrightarrow [\text{dtype. o} = \text{Queue}[T] \wedge \text{otype} = \text{CQueue}[T]$$
$$\wedge \text{Leave. pre}_{\text{Queue}[T]}(\text{currentstate(o)})]\}$$
$$\text{o. Leave}_{\text{CQueue}[T]}()$$
$$\{[\text{oldstate(o). items} = \langle \text{item!} \rangle \frown \text{currentstate(o). items}$$
$$\wedge \text{currentstate(o). size}$$
$$= \#\text{currentstate(o). items} \wedge 0 \leqslant \text{currentstate(o). size} \leqslant \text{max}]$$
$$\Leftrightarrow \text{Join. post}_{\text{Queue}[T]}\}$$

这可以说明，对于子类的不变式扩充但不加强的情况下，子类原本继承父类的操作。虽然子类的不变式扩充了，引进了新的状态变量，但对这类操作是不影响的，我们不必对它进行重新推理。

总结及今后的工作 本文探讨了基于面向对象形式规格说明语言 Object-Z 的多态推理及其重用。根据 Object-Z 不变式、前置条件和后置条件，提出了推理的一般规则，既可以推理一般的行为，也可以推理子类特有的行为。知道了多态变量所指的对象普遍的行为，那么在有些情况下，可以推理它具体所指的对象特有行为时，可以考虑推理的重用。

因为面向对象形式规格说明语言 Object-Z 可以精确地描述大型软件系统，能够对这些形式规格说明进行精确的验证与推理，可以及时地发现需求规格说明中的错误。因此，今后的工作是研究对形式规格说明进行更有效推理的方法。

参 考 文 献

1 Soundarajan N, Fridella S. Behavioral subtyping and behavioral enrichment of multimethods Technology of Object-Oriented Languages and Systems. TOOLS, 2000. 105～114

2 王云峰，李必信，郑国梁. 面向对象 Z 的子类型继承和推理规则. 软件学报，2000，11(4)：481～487

3 Liskov B H. A Behavioral Notion of Subtyping. ACM, 1994, 16(6): 1811～1841

4 Smith G. The Object-Z Specification Language. Advances in Formal Methods. Kluwer Academic Publishers, 2000

5 Sun Jing, Dong Jing Song. Specifying and Reasoning about Generic Architecture in TCOZ, APSEC, 2002

6 Soundarajan N, Fridella S. Inheriting and modifying behavior. Technology of Object-Oriented Languages and Systems, TOOLS, 1997. 148～162

7 Soundarajan N, Fridella S. Reasoning about polymorphic behavior. Technology of Object-Oriented Languages and Systems, TOOLS, 1998. 346～358