基于XML的异构构件组装技术研究

盛贤良，瞿有甜
（浙江师范大学 数理与信息工程学院，浙江 金华 321004）

摘 要：异构构件组装技术已成为基于构件的软件开发中的关键问题。研究现有构件模型及常用组装方法，结合构件组装中的连接子和胶合代码两种已有的方法，搭建一个异构构件的组装模型，并通过XML语言对原子构件组装成模块（复合构件）的整个组装过程进行描述。这种建模方式，有效地屏蔽了构件的异构性，为基于构件的应用系统开发进行了有益的探索，并取得了一定的成效。

关键词：XML；异构构件；构件组装；连接子
中图分类号：TP311 **文献标识码**：A **文章编号**：1673-629X(2008)02-0083-05

Study of Technology for Heterogeneous Component Composition Based on XML

SHENG Xian-liang, QU You-tian
(Coll. of Mathematics, Physics and Information Eng., Zhejiang Normal Univ., Jinhua 321004, China)

Abstract: The technology for heterogeneous component composition has already been the key problem during component based software development. Studies the available component models and methods of component composition at present, designs a model for heterogeneous component composition combining connector with glue code, and describes the whole process of atom-component combining to module (composite component). This method of modeling shields the heterogeneities of components, and establishes certain basis for component based software development.

Key words: XML; heterogeneous components; composition of components; connector

0 引 言

基于构件的软件开发方法（Component Based Software Development，CBSD）是近几年软件工程研究领域的一个热点，CBSD方法通过在软件开发中重用经过验证的构件，提高了软件复用程度、软件开发效率，更好地保证了开发的质量。随着软件构件技术研究的不断深入，一些基于构件开发的支撑平台相继形成。利用商业构件（COTS）、可复用构件开发应用系统已逐渐成为一种趋势。为此，构件技术被认为是解决软件危机的重要手段之一[1]。基于构件的软件开发的重点从程序设计转变为构件组装[2]。

由于缺乏统一的标准，构件技术研究产生了诸多不同的构件模型（如REBOOT、C2、青鸟FLP模型等）和规范（如CORBA、COM/DCOM、EJB等），导致了构件难于互操作和跨平台。一方面，构件模型的异构性，限制了系统开发人员使用来自不同模型的构件；另一方面，运行平台仅支持符合某种特定构件模型的构件，其他模型的构件无法运行。如何将基于异构构件模型的构件进行组装，已经成为CBSD方法研究中一个亟待解决的核心问题。

文中通过对现有构件组装方法的一些优缺点的分析，提出了一个基于XML构件组装描述语言的异构构件组装框架；通过引入连接子（Connector）[3]的概念，对请求一个服务的构件接口和实现这个服务的构件接口进行连接及适配；在组装阶段，依据组装模型及构件实现实体的描述文件，自动生成连接和适配的胶合代码。

1 XML技术概述

XML（eXtensible Markup Language）是SGML（Standard Generalized Markup Language，标准通用标记语言）的一个子集，是简化的SGML[4]。

收稿日期：2007-05-12
基金项目：浙江省科技计划重点科研攻关科技项目（2005C21008）；浙江省自然科学基金（M603245，Y106469）
作者简介：盛贤良（1981-），男，浙江嘉兴人，硕士研究生，研究方向为中间件技术、数据库技术；瞿有甜，副教授，研究方向为软件工程、构件技术和数据库技术。

总体来讲,XML有以下几个特点:

(1) 纯文本:几乎任何工具都可以创建和编辑,使得程序更简单,从而提供了从最小配置文件到企业级数据仓库的可扩展性;

(2) 基于内容的数据标记:可以被不同的程序应用于不同的目的;

(3) 很强的链接能力:可以定义双向链接、多目标链接、扩展链接和两个文档间的链接;

(4) 可格式化:可扩展样式语言可以指定如何显示数据,数据和显示是分离的,可以为同一数据指定不同的样式表用于不同的输出;

(5) 易于处理:对格式的定义严格,具有层次结构,而且与厂商无关。

从数据描述语言的角度看,灵活、可扩展、有良好的结构和约束,有助于理解和代码自动生成。正是由于XML语言在这些方面的优越性,笔者将借助XML语言工具来描述异构构件组装模型。

2 构件模型及其组装方法

2.1 构件定义及其构件模型

构件模型是CBSD方法研究和实践的核心内容,是构件定义和构件性质的具体化。构件模型除了定义构件的本质属性外,还向外界提供其他构件调用的入口服务(Service),同时构件请求外界服务的引用(Reference)结构。因此构件模型可以形式化地定义为:

Def Component: : = < C - Properties, C - Services, C - References, Implementation >

其中:C - Properties表示构件属性集;C - Services表示构件提供的服务集;C - References表示构件需要的引用集;Implementation表示构件本身的实现。

图1给出了构件模型的示意图。

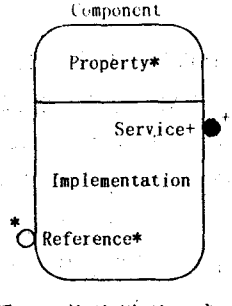

图1 构件模型示意图

从图1可以看出,一个构件有0或多个属性(*号表示),0或多个引用,1或多个服务(+号表示)。每一个服务和引用通过一个接口来实现。

2.2 常用的构件组装方法

构件组装就是在构件模型的基础上研究构件组装机制,包括源代码级组装和基于构件对象互操作运行级的组装等。构件组装的方法主要分为"黑"、"白"、"灰"三种[5]。由于构件组装实现的特性,"灰盒"组装方法是研究和开发的重点。当然,灰盒组装也主要分为四类,分别是:基于框架的,基于体系结构的,基于连接子和基于胶合(Glue)代码。在构件组装技术研究方面,文献[6]中提出了面向应用的两层软构件的调度、管理机制,将不可重用的部分以脚本的形式分离出来,通过改变消息转发部件和脚本部件生成多种框架,解决基于软件总线的软构件系统中的构件组织与管理。北京大学曾在文献[3]中提出的ABC方法是一种基于体系结构的、面向构件的软件开发方法,把体系结构的概念与CBSD方法相结合,把体系结构建模的思想贯穿需求、设计、实现等整个软件开发过程。可以发现,文献[6]中的方法注重通过消息来实现对构件的管理与调度,而ABC方法适用于所有的软件开发过程,两者在异构构件组装方面都没有展开针对性的讨论。

文中在下面的几节中主要论述如何结合连接子和胶合代码,借助XML这种跨平台的描述语言来实现异构构件的组装。

3 基于XML的异构构件组装建模

基于构件的软件开发(CBSD)中,构件组装首先要实现多个(包括一个)原子构件(atom - component)通过某种方式组合在一起构成具有特定功能的模块(Module)——复合构件(composite component)。在复合构件的基础之上进行系统级组装将相对容易,并且与原子构件组装成复合构件的过程极为相似[7],为此,复合构件的组装是构件组装技术中的一个重点和难点。文中将重点对复合构件组装的建模过程和描述方法进行探讨。

结合现有的一些构件组装方法,引入连接子概念以后,可构建如图2所示的模块功能级构件组装模型。

从图2中可以看出,一个构件是否能调用另一个构件是通过这个构件的引用接口和另一个构件的服务接口是否能连接及匹配来决定的,模块之外不能直接设置模块内的构件属性,不能直接调用模块内的构件服务,外界的服务也不能被模块内的构件引用直接调用。模块使得其内的各个构件的构件属性、构件服务和构件引用对外不可见、不可访问,从而实现了对模块内的构件进行封装的目的。此外,外界只能通过模块的入口来访问模块内构件提供的服务,构件对外界的引用也只能通过模块的引用来调用外界服务。

对模块内构件的组装过程主要通过以下五个阶段来实现。

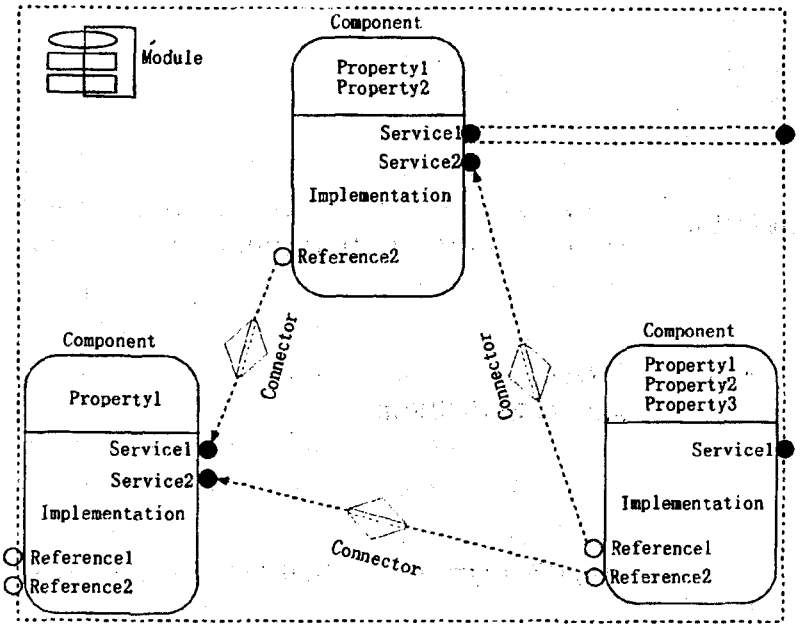

图2　带连接子的模块级构件组装模型

3.1　定义构件类型

构件类型是对构件属性、构件服务和构件引用的定义。把一类可以提供相同服务、需要引用相同服务、具有相同属性的构件,归为一个构件类型。构件类型对于它的构件实例来说,定义了所有可供配置的方面(Aspects)。基于XML的构件类型描述如下:

```
<? xml version="1.0"encoding="utf-8"? >
<componentType xmlns="http://cbsd.zjnu.edu.cn"xmlns:xs=
"http://www.w3.org/XMLSchema"name="构件类型名称">
<! -0或多个types子元素,用于定义描述文件中出现的元素类型->
<types>
  <interface name="接口名称">
    <! -1或多个method元素->
    <method name="方法名称"type="方法返回类型">
      <! -0或多个parameter元素->
      <parameter name="参数名称"type="参数类型"/> *
    </method> +
  </interface> *
</types>
<! -1或多个service元素,表示构件类型对外提供1或多个服务->
<service name="服务名称">
<! -每一个服务提供一个接口,描述调用标准和规范->
<interface name="接口名称"/>
</service> +
<!  0或多个reference元素,表示构件类型需要引用0或多个外界服务->
<reference name="引用名称"multiplicity="0..1 or 1..1 or 0..n or 1..n">
<! -每一个引用需要一个接口->
<interface name="接口名称"/>
</reference> *
<! -0或多个property元素->
<property name="属性名称"type="属性类型"default="属性默认值(可选)"many="是否为多值属性(True/False)"required="是否为必选属性(True/False)"/> *
</componentType>
```

其中reference元素的multiplicity属性是可选属性,它有四个待选Value,分别是"0..1"、"1..1"、"0..n"、"1..n"。在构件实例中,"0..1"表示该构件最多只能与一个本构件之外服务连接;"1..1"表示该构件与且仅与一个本构件之外服务连接,这是该属性的默认值;"0..n"表示该构件可与0个或多个本构件之外服务连接;"1..n"表示该构件至少需要与一个本构件之外服务连接。

3.2　对构件进行实例配置

经过对构件类型的定义,得到了关于待组装的构件的所有构件类型的XML描述文件。对定义好的构件类型进行实例配置就是对每一个构件类型的构件属性和构件引用进行配置。

构件实例配置主要分为以下两步:

1) 给构件属性赋值,尤其是构件类型定义中property元素的required属性为true的属性赋初值。

2) 把构件引用连接到目标服务,必须把构件的非空引用连接到相应的目标服务上。构件的非空引用就是构件类型定义中reference元素的multiplicity属性为1..1或者1..n的引用(包括没有multiplicity属性的情况,因为默认是1..1)。

但在做第二步的过程中会发现,由于各构件类型可能存在异构性,调用标准和规范不一致,导致构件引用连接到目标服务器无法实现的情况,为此引入了图2中的连接子(Connector),通过连接子来实现引用与服务之间的连接与适配,同时还要做一些兼容性的匹配工作。兼容性的匹配主要有以下三个层次:

(1) 首先在逻辑上,两个接口对应的方法功能必须一致,这个判断可能需要用户的参与;

(2) 其次在方法描述上,引用接口和服务接口的参数个数和返回类型必须一致,引用接口方法中的参数必须在服务接口的方法中找到兼容(参数类型一致)的参数;

(3) 在异常处理上,服务接口方法的每一个异常处理在引用接口方法上必须有一致或兼容(异常类型一致)的异常处理。

连接器有两个端口，一个是引用端口(Reference Port)，另一个是服务端口(Service Port)[8]。顾名思义，连接器的引用端口与构件引用接口连接，服务端口与构件服务接口连接。连接器适配功能体现在接口的逻辑功能，方法描述及异常处理上进行匹配。通过构件实例配置，产生一个模块(Module)，供应用系统组装和部署时使用。

构件实例配置的 XML 描述如下：

```
<? xml version="1.0"encoding="utf-8" ? >
<module xmlns:xs="http://www.w3.org/XMLSchema"xmlns:
v="http://cbsd.zjnu.edu.cn"name="模块名称">
<! -每一模块(Module)至少有 1 个构件.0 或多个连接子
(Connector)- >
<component name="构件名称"type="构件类型">
<implementation/>
  <! -0 或多个 v:构件属性名称元素,构件属性名称已在
该构件的构件类型属性中定义- >
  <v:构件属性名称 modulePropertyName="是否为 Module
属性,如果是则该名称是 Module 属性的名称">
属性值
  </v:构件属性名称> *
  <! -0 或 1 个 references 元素- >
  <references>
    <! -1 或多个 v:构件引用名称 子元素,构件引用名称
已在该构件的构件类型引用中定义- >
    <v:构件引用名称 moduleReferenceName="是否为 Mod-
ule 引用,如果是则该名称是 Module 引用的名称">
  构件名称/服务名称
  </v:构件引用名称> +
  </references>?
</component> +
<connector>
  <! -连接子的连接起点的构件名和引用名- >
  <source>构件名/引用名</source>
  <! -连接子的连接终点的构件名和服务名- >
  <target>构件名/服务名</target>
  <! -method-mappings 子元素可选,当没有该元素是,代
表两个接口的所有方法完全一致,不需要连接子- >
  <method-mappings>
    <! -1 个到多个 method-mapping 子元素,用来描述每
个方法的匹配情况- >
    <method-mapping source="引用接口方法名"target=
"服务接口方法名"type="方法返回类型">
    <! -0 或 1 个可选子元素 param-mappings,描述每个
参数的匹配情况- >
    <param-mappings>
      <param-mapping source="引用接口方法中的参数
名"target="服务接口方法中的参数名"type="参数类型"/> +
    </param-mappings>?
  </method-mapping> +
  </method-mappings>?
</connector> *
</module>
```

配置构件实例是单个原子构件组装为复合构件的关键，在配置构件实例中不仅仅设置了构件及模块的属性值，更重要的是配置构件之间的关系，建立了服务请求者和服务提供者之间的连接。通过连接子来连接构件的引用接口和服务接口，并匹配接口的功能、方法描述(包括参数)及异常处理等。

3.3 指定构件实现体

配置构件实例后，构件还是一个概念上的构件，没有对应的构件实现体。通过指定操作之后，构件才成为一个既有概念、又有实现的实体。实现一个概念构件的具体技术可以有很多，比如 java 和 web services 等。

在配置构件实例的描述中，指定构件实体在 implementation 元素中描述，比如将上述描述中<implementation/>改为<implementation.java class="implementation-class"/>或<implementation.ws wsdl="implementation-wsdl"/>，即分别表示用 Java 和 Web Service 技术来实现。

3.4 进行模块属性、服务及依赖引用的绑定

模块作为一种复合构件，可类似于构件形式化定义模块(Module)：

Def Module::=<M-Properties, M-Services, M-References>

其中，M-Properties 表示模块的属性集，M-Services 表示模块对外提供的服务集，M-References 表示模块需要的引用集。

可以发现，模块的定义与构件的定义极为相似，只是缺少了实现，因为模块的实现是依赖于构件及连接子的实现。模块属性来自于模块中原子构件的属性；模块服务来自于原子构件的服务，是模块的入口，表示模块可以对外提供的服务；模块引用来自于对其他模块或构件的依赖，也来自与原子构件的依赖。这阶段主要的工作是对模块中原子构件的属性、服务、引用与模块的属性、服务以及引用进行一对一的绑定。下面给出的模块与原子构件绑定方案的 XML 描述：

```
<? xml version="1.0"encoding="utf-8" ? >
<module xmlns="http://cbsd.zjnu.edu.cn"xmlns:xs="http://
www.w3.org/XMLSchema" name="模块名称">
<component/> +
<connector/> *
<! -0 或多个模块属性- >
<module-property name="模块属性名称">
```

<!--一个强制子元素 binding,其中属性名称为原子构件属性的 modulePropertyName 值-->
<binding>构件名称/属性名称</binding>
</module-property> *
<!--1 或多个模块服务-->
<module-service name="模块服务名称">
<!--一个强制子元素 binding,其中服务名称为原子构件服务名称-->
<binding>构件名称/服务名称</binding>
</module-service> +
<!--0 或多个模块引用-->
<module-reference name="模块引用名称">
<!--一个强制子元素 binding,其中引用名称为原子构件引用的 moduleReferenceName 值-->
<binding>构件名称/引用名称</binding>
</module-reference> *
</module>

3.5 生成胶合代码

构件所引用的服务和所提供的服务都通过接口来描述。通过连接构件的引用接口与另一个构件的服务接口来实现调用。事实上,构件的实现实体可能是不一样的,并且在接口规范和实现体上可能存在很大的异构性,如有的采用 EJB,而有的采用 Web Services,但它们可能可以提供同样的功能,只是描述服务的接口可能稍有不同,比如接口方法名称和参数顺序等。生成胶合代码就是在系统组装人员完成组装关系建模之后,生成每一个存在连接关系的构件通过引用接口调用目标构件服务接口的胶合代码。胶合代码匹配两个接口的方法、参数;如果连接两端构件的类型不同,胶合代码则包括适配构件类型。因为在生成胶合代码之前,对组装模型已有详细的 XML 描述文档,则胶合代码的实现可以自动完成。通过适配接口和适配构件类型,胶合代码实现了构件实例之间的调用。可见,生成胶合代码归根结底就是生成每个连接的连接子代码。可以发现,连接子代码主要有上述三个作用:连接接口、适配接口、适配构件类型。

4 结束语

提取了"灰"盒构件组装技术中的两类方法,根据各构件模型之间的差异,结合连接子和胶合代码技术,对异构构件组装进行了建模,并通过 XML 语言对原子构件组装成模块(复合构件)进行了详细描述。该模型屏蔽了异构构件存在的差异,使得用户对异构构件可进行透明访问与使用。文中所做的工作只是构件组装技术领域的一小块,接下来的研究和实践工作还有很多,如在构件组装的过程中如何尽量避免人的干预,如何进一步提高胶合代码生成的自动化程度以及如何扩大对构件异构性的支持都是进一步要考虑和研究的内容。

参考文献:

[1] NATO CO-5957-ADA. NATO Standard for Management of a Reusable Software Component Library[C]. Tokyo: NATO Communications and Information System Agency, 1991: 32-43.

[2] Clements P C. From subroutines to subsystems: component-based software development[C]// In: Brown A W, ed. Component-Based Software Engineering: Selected Papers from the Software Engineering Institute. Los Alamitos, CA: IEEE Computer Society Press, 1996:3-6.

[3] 梅　宏,陈　锋,冯耀东,等. ABC:基于体系结构、面向构件的软件开发方法[J]. 软件学报,2003,14:721-732.

[4] 印　旻,景建萍. XML 基础与应用教程[M]. 北京:清华大学出版社,2001:10-11.

[5] 王至坚,费玉奎,娄洲清. 软件构件技术及其应用[M]. 北京:科学出版社,2005:123-170.

[6] 白　涛. 一种基于软件总线可重用构件框架的设计与实现[J]. 微电子学与计算机, 2005,22(1):47-49.

[7] Taylor R N, Dashofy E M. The Use of Middleware to Implement Connectors in Distributed Software Architectures[EB/OL]. 1999. http://www.ucop.edu/research/micro/97_98/97_177.pdf, 1999:97-177.

[8] 叶　菲. 一个基于 Java 的面向 Web 应用的构件组装工具[D]. 上海:复旦大学,2006.

(上接第 82 页)

[2] 刘同明,夏祖勋,解洪成. 数据融合技术及其应用[M]. 北京:国防工业出版社,1998:1-13.

[3] 涂国平,邓群钊. 多传感器数据的统计融合方法[J]. 传感器技术,2001,20(3):28-30.

[4] Lou R C, Lin M, Scherp P S. Dynamic multi-sensor data fusion system for intelligent robots[J]. IEEE Journal of Robotics and Automation, 1988,4(4):386-396.

[5] 杨宾峰,罗飞路. 一种改进的一致性多传感器数据融合算法[J]. 理论与实践(计量技术),2006.5:3-5.

[6] 陈福增. 多传感器数据融合的数学方法[J]. 数学的实践与认识,1995,25(2):13-16.

[7] 王　威,周军红,王润生. 多传感器数据融合的一种方法[J]. 传感器技术,2003,22(9):39-41.

[8] 涂国平,叶素萍. 一种传感器数据的融合算法[J]. 传感器技术,2003,22(3):30-32.

[9] 蒋正新,施国梁. 矩阵理论及其应用[M]. 北京:北京航空学院出版社,1998:371-378.