面向服务的构件规约

唐龙业 王 宇 王志坚
(河海大学计算机与信息学院 南京 210098)

摘 要 目前已有构件规约大多是面向接口方法的型构和行为描述,其主要不足是缺乏针对业务逻辑的完整性描述,而这种规约描述的脆弱性直接影响到构件复用的效率。在需求分析和获取基础上给出构件原子服务、服务的定义及其描述方法;然后在此基础上制定面向服务的构件规约,它是对目前方法级构件规约的扩展;最后从CBD过程层面分析讨论了规约的具体应用。

关键词 构件规约,原子服务,服务组合,CBD过程
中图法分类号 TP311 **文献标识码** A

Service-oriented Component Specification

TANG Long-ye WANG Yu WANG Zhi-jian
(College of Computer & Information, Hohai University, Nanjing 210098, China)

Abstract Most of existing component specifications are focused on describing method-oriented interface signatures and behaviors, but their lack of full description of business logic has influenced the efficiency of component reuse directly. On the basis of requirement analysis and elicitation, atomic services and services were defined and the corresponding methods to describe them were given respectively. And then, service-oriented component specifications (SOCS) were defined and the detailed domain application samples were given. SOCS is not a substitute for existed component specifications but an extended version of them. Finally, this paper talked about its application in the level of CBD process.

Keywords Component specification, Atomic service, Service composition, CBD process

制定构件规约的基本要求是保证信息的充分性[1],即其应该尽可能满足构件复用的需要。开发者通过规约详细描述接口中的显式数据类型和结构(接口型构规约)、功能实现方法及约束(行为规约)。构件规约包含构件多角度特征的描述,其内容和形式对于构件的检索/适配、测试、组装等均有重要影响。所以,制定构件规约不仅考虑构件自身信息的充分描述,更应该从CBSE层面考虑其应用,以提高CBD过程的效率。

1 相关研究

目前,依据描述内容的不同,主要有3种规约形式:接口规约(Interface Specification)、行为规约(Behavior Specification)和非功能性规约(Non-function Specification)。本文主要研究前面两种。

接口规约是关于构件接口的语法描述,如接口名称、类型、属性等,提供构件复用者理解和使用构件的最基本信息。在接口规约中,一个构件提供一组被命名的接口(或类型),每个接口命名一组操作,每个操作有零个或多个输入输出参数及相关联的语法规约。接口规约的典型例子是CCM(CORBA Component Model)中对构件的描述,CCM为CORBA构件定义多种接口[2]。但是接口规约缺乏对构件内部逻辑的描述,所以,其对构件的描述信息相对单一。

行为规约是对构件操作语义的描述,通常包括实例不变式、操作的前置和后置条件、一般断言和循环不变式等[3]。Michihiro[4]利用一个代数行为规约Projection-style描述构件行为,其包括型构和通信规约两个部分,分别用于描述构件的树结构,如事件模式、静态结构和动态结构。Eriksson[5]和Filipe[6]则利用UML协作图、序列图和状态图描述构件的行为。但是,这些规约或者仅基于方法层级描述,或者缺乏直观易理解的表达形式等而存在实际应用上的困难。

青鸟模型[7]中构件规约JB-CDL包含9个部分,即功能移出、模板参数、特例化实例化描述、协作规约、成员规约、规约互联、协作对象、成员对象和对象互联。但是,其构件粒度及规约内容需要业务逻辑的体系结构建模和高层描述的指导。周[8]给出了基于领域问题划分的规约语义,但是其缺乏领域之间的语义普适性(如不同领域间特征的差异)研究。

OSGi提出面向服务的构件模型,旨在通过服务实现更灵活的构件动态配置和部署[9,10]。但是,该模型中的构件仅是作为软件系统架构层次的一个概念,而不是通常意义上可复用的功能构件。OSGi中的服务沿用现有的Web Service概念,并且没有给出服务的具体描述形式。

孟等[11]提出的服务规约概念与本文有类似之处。但是,

到稿日期:2012-04-21 返修日期:2012-07-05 本文受国家自然科学基金(61103017)资助。
唐龙业(1973-),男,博士,主要研究方向为软件构件技术、分布式计算等,E-mail:tly@hhu.edu.cn;王 宇(1979-),男,博士,主要研究方向为分布式计算等;王志坚(1958-),男,教授,博士生导师,主要研究方向为软件构件技术、分布式计算等。

与本文研究不同的是，其服务规约采用了基于自动机中状态转换的行为表达形式，并通过实施行为片段(状态和操作构成的转换序列)匹配实现大粒度业务构件的提取。

通过上述总结看出，现有构件规约更多是方法粒度的型构及行为(操作前后置条件)信息描述，而缺少针对复用所需要的业务逻辑(接口服务)的完整、直观描述，因此降低了增强构件可测试性方法的有效性和针对性，进而影响了构件复用的工作效率。

本文主要在已有构件规约基础上，提出基于原子需求的构件原子服务定义，实现接口方法之间逻辑依赖关系的直观表达；在此基础上给出构件服务的定义和描述形式，并制定面向服务的构件规约，进而提供了一种开发者和复用者可共享的构件接口视图——服务，以提高构件的可理解性和CBD的过程效率。

2 基本定义

2.1 构件设计

需求反映领域中的特定业务逻辑或者从复用者角度对软件要提供服务的要求。而服务则是从满足复用者需求角度描述软件行为(特定的业务逻辑及相应的输入/输出)，是设计结果的体现。

一项软件的服务总是对应一个包含特定输入输出数据的操作序列，其反映最终用户的需求或目标。所以，本文把需求表示为反映其逻辑过程和特征的“输入输出”数据集合，记作$(InitState,\{\langle IN_i,OUT_i\rangle\})(i\leqslant C(\text{常数}))$，其中：

• $InitState$ 表示一个需求所对应的逻辑过程的初始逻辑状态；

• $\langle IN_i,OUT_i\rangle$：$IN_i$ 表示得到 OUT_i 的一个或一组输入数据，可以是操作、对象、文档及具体数据等显式数据形式，可以为空；OUT_i 代表对应 IN_i 的输出数据，它表示业务逻辑(即需求)要达到的一个直接目标：一个或一组数据、图表、文档或者对象等任何显式的数据形式。

图1为“用水申报”业务需求分解树。

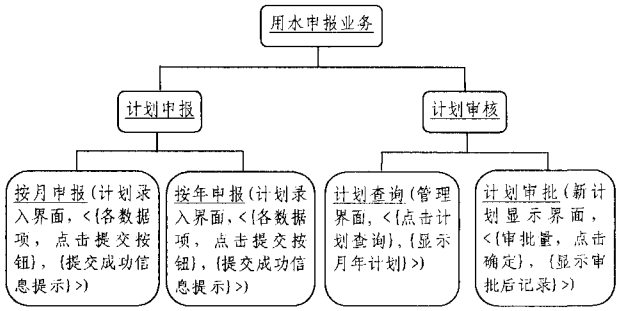

图1 “用水申报”业务需求

定义1(原子需求，atomic requirement，AR) 一个有限的输入输出$(InitState,\langle IN_i,OUT_i\rangle)$称作原子需求，如果在初始状态 $InitState$ 下有输入 IN_i(可以为空)，则 OUT_i 是 IN_i 对应的显式输出结果，并且该结果是需求要实现的一个具体目标。

术语“原子”用于强调$(InitState,\langle IN_i,OUT_i\rangle)$的不可再分性，它构成需求表达和描述的最小概念单位，反映一个最小粒度的实现目标。而从业务逻辑角度，一个原子需求具有唯一的业务特征标识，通过该特征标识能够界定一个特定的业务操作过程，并且实现原子需求之间的相互区分。

定义2(需求，requirement，R) 一个需求是一个有限的输入输出集合$(InitState,\{\langle IN_i,OUT_i\rangle\})(1<i\leqslant C(\text{常数}))$。

与定义1相比较，需求包含多个要实现的原子目标。所以，定义2说明：从业务逻辑角度，一个需求 R 包含一个原子需求集合，记作 AR_set；并且 R 可以由集合中的原子需求按照某种方式组合得到，即本文中定义的需求具有组合特性。另外，$InitState$ 通常是集合中第一个原子需求对应的初始状态描述。

本文从以下两个方面理解定义2：

(1)对任意一个需求 R，通过分解总能得到一个与其对应的原子需求集合 $AR\_set=\{AR_1,AR_2,\cdots,AR_m\}$($m$ 是一个常数)；

(2)一个需求 R 可以由相应的若干原子需求按照某种方式组合得到。

对于(1)的原子需求集合，给出以下求解方法：

由定义1和定义2知，任意一个需求 R 对应着显式输入输出集合$(InitState,\{\langle IN_i,OUT_i\rangle\})(1\leqslant i\leqslant C,C$ 是常数)，其中 $InitState$ 表示 R 的初始逻辑状态；IN_i 可以为空；OUT_i 代表 R 要实现的第 i 个直接目标。对于$\{\langle IN_i,OUT_i\rangle\}$，有：

(1)如果 $i=1$，即 R 对应一个唯一的实现目标，则根据定义2有，$(InitState,\{\langle IN_1,OUT_1\rangle\})$是原子需求，即需求 R 对应集合$(InitState,\{\langle IN_1,OUT_1\rangle\})$，这说明 R 本身是一个原子需求。

(2)如果该集合中 $1<i\leqslant C$(常数)，则有如下操作步骤：

Step1 根据定义2，确定首个原子需求即 $AR_1=(InitState,\langle IN_1,OUT_1\rangle)$。

Step2 对第二个输入/输出对$\langle IN_2,OUT_2\rangle$，有如下两种情况：

a)如果 OUT_2 依赖于 OUT_1(满足偏序关系)，即 $OUT_1\cap IN_2\neq\varnothing$(空)，则在 AR_1 基础上确定第二个原子需求 $AR_2=(InitState,\langle IN_1,OUT_1\rangle,\langle IN_2,OUT_2\rangle)$；

b)如果 OUT_2 不依赖于 OUT_1(不满足偏序关系)，即 $OUT_1\cap IN_2=\varnothing$(空)，则确定第二个原子需求即 $AR_2=(InitState,\langle IN_2,OUT_2\rangle)$。

Step3 对任一个输入/输出对$\langle IN_i,OUT_i\rangle(3\leqslant i<C)$，重复Step2，直到最后一个$\langle IN_C,OUT_C\rangle$依照相关过程判断并处理完毕。

Step4 得到原子需求集合$\{AR_1,AR_2,\cdots,AR_m\}$。

完毕。

对于(2)中的原子需求组合问题，上述求解方法Step2中已经给出两种组合方式。下面以定义的形式给出原子需求的组合方式。

定义3 串行组合(;)和并行组合(‖)。

对任意两个原子需求 $AR_i=(InitState,\langle IN_i,OUT_i\rangle)$和$AR_j=(InitState,\langle IN_j,OUT_j\rangle)$：

$AR_i;AR_j$：称作串行组合，如果 $OUT_i\cap IN_j\neq\varnothing$ 即 AR_j 的实现依赖于 AR_i 的实现。

$AR_i\parallel AR_j$：称作并行组合，如果 $OUT_i\cap IN_j=\varnothing$ 即 AR_j 的实现与 AR_i 的实现之间不存在依赖关系。

定义1－定义3说明，一个需求可以由相应的若干原子需求通过某种组合形式得到。而需求定义体现了业务逻辑的高内聚性，这也是确认构件功能粒度(即实现哪一些原子需求)以及实施构件设计的基础。

图 2 是“用水申报”业务构件实现。

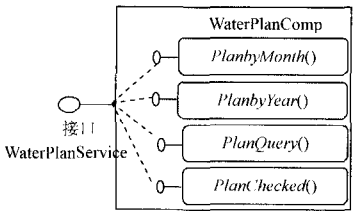

图 2 “用水申报”业务构件设计

2.2 原子服务

定义 4(原子服务,atomic service,AS) 指一项原子需求的构件接口实现。

一项原子需求对应一个特定的输入与输出对〈IN,OUT〉,所以,原子服务实现对应〈IN,OUT〉的一个特定业务操作过程。

定义 4 从接口实现角度给出对应业务操作过程(原子服务)的定义。构件的接口实现表现为若干方法的组合,而组合是若干方法协作实现一项原子服务的方式。其中,单个接口方法是原子服务的最小描述粒度。对于方法组合形式,给出下列组合表达式。

定义 5(方法组合表达式,function assembly expression)一个组合表达式由接口方法和若干组合运算符构成。其中:

(1)方法型构:$f_i(InType,OutType)$,$InType$ 表示接收数据的类型,可以为空;$OutType$ 表示返回或输出数据类型。为了书写方便,可以省略两个类型声明。

(2)组合运算符:对任意接口方法 f_i,f_j 和 f_k,

- 调用[]:$f_i[f_j]$表示在 f_i 中调用 f_j。
- 顺序;:$f_i;f_j$ 表示执行完 f_i 后再调用执行 f_j。
- 并行|:$f_i|f_j$ 表示 f_i 和 f_j 之间条件(如 IF_ELSE)或无条件(事件驱动)组合。
- []、;、|的嵌套组合:如 $f_i;f_j[f_k]$表示先执行 f_i,然后执行 f_j 并在 f_j 中调用 f_k。

图 2 所示“用水申报”构件设计所提供的原子服务是:AS_1:PlanbyMonth(),AS_2:PlanbyYear(),AS_3:PlanQuery(),AS_4:PlanQuery();PlanChecked()。

定义 6(服务,service,S) 指满足一项需求的构件接口实现。

根据定义 1、定义 2 和定义 4 有,一个服务包含一个由若干原子服务构成的最小服务集合$\{AS_1,AS_2,\cdots,AS_N\}$($N$ 是常数)。一个服务或者是一个原子服务,或者是若干原子服务的某种组合。所以,服务对应一个满足特定需求的若干个输入输出对构成的集合:$\{AS_1:(IN_1,OUT_1),AS_2:(IN_2,OUT_2),\cdots,AS_N:(IN_m,OUT_m)\}$($m$ 为常数)。

原子服务构成服务的最小描述概念单位,反映最小粒度的业务处理过程并产生满足特定原子需求的输出,而其对应的业务处理过程在执行上具有相对的独立性。所以,给出如下原子服务的组合定义和性质。

性质 1 原子服务组合表现为事件驱动方式组合。

该性质的含义是:假定一个服务 S 包含最小粒度的原子服务集合$\{AS_1,AS_2,\cdots,AS_N$(N 为常数)$\}$。那么,在任一个原子服务不是其它原子服务串前缀的情况下,S 的实现方式有 $N!$(N 个原子服务的个数)种,即任意原子服务的先后被执行顺序都是合法的。但是,原子服务组合要符合业务逻辑。

在性质 1 基础上,给出如下原子服务循环组合定义。

定义 7(原子服务循环组合) 任一项原子服务可以在一次服务执行过程中多次被执行,记作符号 *。

依据性质 1 和定义 7,原子服务组合的服务描述形式表示为:

$\{[AS_1]^* \| [AS_2]^* \| \cdots \| [AS_N]^*\}$($N$ 为原子服务个数)

式中,“*”表示一个原子服务在一次服务执行中可多次出现(循环次数取决于用户的任务规模),“‖”表示序列中的原子服务之间的并列组合关系。

2.3 构件服务的行为语义表达

构件服务反映特定业务逻辑过程的接口实现,在一定程度上也体现构件之间的接口交互模式。

但是,交互模式并不是构件之间的组装方式,前者是指不同构件之间直接的消息传递方式;而后者是应用层级构件之间的组合,如并行、选择、复制、顺序、中断和连接等组装方式,这种组装不一定发生构件之间直接的消息传递事件。

根据定义 4、定义 5,构件服务也是一个方法序列,该序列构成各相关对象之间的一个交互消息序列。所以,构件服务可以转换为具有等价行为语义的 UML 顺序图。具体转换过程如下:

Step1 根据一个服务对应的方法组合表达式确定参与交互的构件实例(集合),因为一个方法关联源和目的两个实例对象(也可以是自调用);

Step2 确认每一个构件实例在顺序图中从左往右的排列顺序;

Step3 按照从左往右的顺序,依次把表达式中每一个方法标注为相关联构件实例之间的消息。

依据上述转换过程可以实现构件服务到语义等价的 UML 顺序图的转换。图 3 示出“计划审批”服务对应的 UML 顺序。

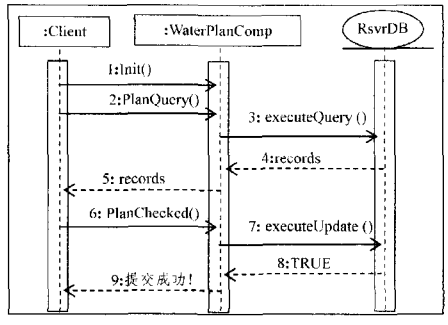

图 3 “计划审批”业务顺序图

构件服务表达的交互行为可以用 UML 顺序图等价表示,并可以在两者之间实现语义等价转换。而原子服务能够反映接口方法之间的依赖关系及构件接口可能的交互行为语义特征。所以,本文有关构件服务的描述方法合理且可行。

3 面向服务的构件规约

本文中的构件规约包括接口型构规约和面向服务的行为规约两部分。前者是关于构件接口的语法规范描述,而后者是针对构件的功能性特征说明。

3.1 基本要素

规约中涉及的基本要素(术语)包括:

- 属性(atribute):反映特定问题域的数据参数,包括类型、取值范围等。此处主要是指接口可视或可配置的公共类

型参数。

• 方法(method):完成特定处理过程的功能实现单元,包括名称、类型及(输入/出)形式参数。

• 约束(constraint):布尔类型条件表达式。常见形式是前置、后置条件和(类)不变式。

• 原子服务(atomic service):业务逻辑的最小描述单位,也是构件接口的最小服务单位。具有唯一的标识(特定的输入/出对),以区别于其它原子服务。

• 构件服务(component service):若干原子服务的一个组合序列。

• 构件(component):以 OO 类为实现体的封装,可独立部署单元,在组装系统中具有唯一的构件标识。型态有基本和复合构件两种。

• 接口(interface):构件对外可视窗口,构件通过接口对外提供服务。通常包括型构、功能性和非功能性规约。

构件接口的描述形式如图 4 所示。其中,各基本要素的标签符号含义解释如下:

(1)INTERFACE(行 1):标注接口名字,并且每一个接口对应一个 END INTERFACE(行 18)作为结束标注符号。

(2)IMPORT(行 2):引入一个关联接口以声明需端口方法所在接口,或者声明同一构件的另一个接口以描述不同接口之间的关联或者某种依赖关系。

(3)TYPE(行 3):声明特定数据类型,是可选项。

(4)型构规约部分(行 5—行 10):包括构件接口中可定制的数据参数及接口方法声明。前者在 DATA_COSTOMIZED_SECTION(行 5)和 END DATA_ COSTOMIZED_SECTION(行 7)之间描述,而后者在 PORT_SECTION(行 8)和 END PORT_SECTION(行 10)之间描述。

(5)行为规约(行 12—行 17):进行原子服务的声明(在 ATOMIC_SERVICE_ SECTION 和 END SERVICE_ SECTION 之间)和原子服务组合序列描述(在 AS_COM POSITION_PATTERN 和 END AS_COMPOSITION_ SECTION 之间)。

```
1. INTERFACE 接口名称//一个构件接口
2 .[IMPORT 接口]//可以引入其它接口
3.[TYPE 类型名字] //用于声明数据类型
4.// 型构规约部分(行 5—行 10)
5. DATA_COSTOMIZED_SECTION //可定制的数据参数
6.…
7. END DATA_COSTOMIZED_SECTION
8. PORT_SECTION //接口方法声明
9.…
10. END PORT_SECTION
11.// 行为规约部分(行 12—行 15)
12. ATOMIC_SERVICE_SECTION //原子服务声明
13.…
14. END ATOMIC_SERVICE_SECTION
15. AS_COMPOSITION_PATTERN //原子服务组合形式
16.…
17. END AS_COMPOSITION_SECTION
18. END INTERFACE
```

图 4 构件接口表示

3.2 接口型构规约

该规约主要针对构件接口中方法的语法规范及接口中可定制的数据参数实施描述。其中:参数的类型可以是基本类型或引用类型,也可以是结构类数据类型(如数据库中表数据),而结构类参数可以在 STRUCTURED_DATA 中声明。图 5 所示为"用水申报业务"构件 WaterPlanComp 对应的型构规约。

```
1. INTERFACE WaterSchedService
2. DATA_CUSTOMIZED_SECTION
3. String RsvrInfoDB //水量信息数据库
4. Int MAXBYYEAR=500//最大年申报水量(万立方)
5. Int MAXBYMONTH=50//最大月申报水量(万立方)
6. END DATA_CUSTOMIZED_SECTION
7. PORT_SECTION
8. void PlanbyMonth(OUT:Boolean result)//提交月用水计划
9. {@pre:申报数据小于 MAXBYMONTH)
10. @Post: AND result=TRUE AND 数据存入数据库
11.      OR result=FALSE}
12. void PlanbyYear(OUT:Boolean result) //提交年用水计划
13. {@pre:申报数据小于 MAXBYYEAR
14. @Post: result=TRUE AND 数据保存 OR result=FALSE}
15. void PlanQuery(OUT:Recordset rs)//查询输出新计划
16. {@pre:会话用户为超级用户
17. @Post:rs!=NULL OR rs=NULL}
18. void PlanChecked(IN:String id; OUT:Boolean result)
19. {@pre:id!=NULL//计划号
20. @Post:(数据小于 MAXBYYEAR 或 MAXBYMONTH)
21. AND result=TRUE AND 数据保存 OR result=FALSE}
22. END PORT_SECTION
23. END INTERFACE
```

图 5 构件 WaterPlanComp 型构规约

3.3 构件行为规约

行为规约是关于构件业务逻辑的说明。原子服务是构件可执行的独立功能单位,其反映方法之间的依赖关系。构件服务表现为原子服务之间的任意有效组合形式。所以,本文构件行为规约是针对原子服务和原子服务组合形式的规约。规约形式如图 6 所示,示例构件 WaterPlanComp 的服务规约如图 7(面向一般用户)和图 8(面向上级主管部门用户)所示。

```
1. ATOMIC_SERVICE_SECTION //(原子)服务描述
2. 原子服务标识{
3.    方法组合表达式
4.        @Pre:
5.        @Post:
6.        [Invarint:]
7.        }
8.…
9. END ATOMIC_SERVICE_SECTION
10. AS_COMPOSITION_PATTERN//原子服务组合形式
11.    原子服务组合表达式
12. END AS_COMPOSITION_SECTION
```

图 6 构件服务规约形式

```
1. ATOMIC_SERVICE_SECTION
2. AS1{PlanbyMonth()
3. @Pre:显示系统管理页面并点击【计划申报】
4. @Post:月计划保存或"提交不成功"提示页面
5. Invarint:":WaterPlanComp"存在并且 RsvrInfoDB 打开}
6. AS2{PlanbyYear()
7. @Pre:显示系统管理页面点击【计划申报】
```

8. @Post:年计划保存或"提交不成功"提示页面
9. Invarint:":WaterPlanComp"存在并且 RsvrInfoDB 打开}
10. END ATOMIC_SERVICE_SECTION
11. AS_COMPOSITION_PATTERN
12. [AS1]* ‖ [AS2]*
13. END AS_COMPOSITION_PATTERN

图 7 构件 WaterPlanComp 面向用水单位的服务规约

1. ATOMIC_SERVICE_SECTION
2. AS3{PlanQuery()
3. @Pre:显示系统管理页面并点击【计划审批】
4. @Post:显示待审批的新用水计划
5. Invarint:":WaterPlanComp"存在并且 RsvrInfoDB 打开}
6. AS4{PlanQuery();PlanChecked()
7. @Pre:点击新计划后的【审批】链接
8. @Post:审批记录保存或者"提交不成功"提示页面
9. Invarint:" :WaterPlanComp"存在并且 RsvrInfoDB 打开}
10. END ATOMIC_SERVICE_SECTION
11. AS_COMPOSITION_PATTERN
12. [AS3]* ‖ [AS4]*
13. END AS_COMPOSITION_PATTERN

图 8 构件 WaterPlanComp 面向上级部门的服务规约

特别说明:图 8 中 AS3、AS4 实现的用水计划审批业务与图 7 中 AS1、AS2 实现的(月、年)用水计划申报的使用权限不同。前者需要上级主管部门用户(超级用户)身份,而后者是用水单位身份(一般用户)。所以,在使用操作前需要判断当前会话用户的身份,即是一般用户还是超级用户。其具体实现并不复杂,只要在用户系统登录后保存当前会话用户身份类别,当调用这些方法时判断其用户类别即可。相关判断验证皆可以在复用者粘合代码中完成,而身份识别在方法外部由复用者根据需要设定可以增强构件复用的灵活性。

4 基于 CBD 过程的规约应用考虑

4.1 构件检索与适配

根据面向服务的构件规约 SOCS 很容易创建基于特征值的检索树(可以构建与图 1 所示需求分解树类似的特征树)。这些特征词体现构件要实现的业务逻辑,并且与图 6—图 8 所示服务规约中功能逻辑描述保持一致。依据这些特征词,可以实现目前应用较广泛的基于关键字、基于刻面特征等的构件检索及适配策略,即在此基础上可以实施基于特征词的粗粒度构件检索工作,如针对构件 WaterPlan Comp 的构件特征词:用水审批-用户计划-(年月)申报、审批。

另外,结合服务规约能够给出每一个特征词所关联的服务序列(确定的输入输出或操作序列),进而可以实施更准确的细粒度的领域构件检索与适配[8,11]。

4.2 构件元数据

开发者提供何种构件元数据信息及所提供信息的有效性成为能否增强构件可理解性的关键。

原子服务及其组合形式反映构件的功能逻辑实现,所以,开发者可以针对构件接口提供的原子服务实施测试并提供相应的测试元数据即测试用例。把原子服务作为提供相关元数据的基准单位,使得对元数据的有效性度量更容易实现。对于复用者,可以首先基于业务需求产生测试用例集,然后把测试用例集与开发者提供的元测试用例集合进行比较分析,以进一步深入准确地评估构件的业务逻辑是否满足复用需求。同时,原子服务及其组合特性也使得基于原子服务产生的测试用例具有服务层级的组合特性。

4.3 构件组装

原子服务表达构件接口中最小的功能单位及可能的方法间的逻辑依赖关系。所以,原子服务及其组合形式体现构件的设计逻辑和接口行为特征,特别对于领域业务构件,原子服务的业务逻辑和行为特征更加明显。

如图 8 所示,构件 WaterPlanComp 服务规约中原子服务 AS4(对应方法序列 PlanQuery();PlanChecked())的表达逻辑关系是:首先通过调用方法 PlanQuery()查询是否有新的用水计划,如果有新的计划,则对查询到的每一项新用水计划通过调用 PlanChecked()完成审批。这一过程也体现复用者相关设计页面与构件接口的组装逻辑:两个方法按照先后顺序被调用。规约中的相关约束虽然采用自然语言形式,但是均给出行为实施前后的系统状态,如显示或产生哪一些数据、界面等。而这些对(特别是领域)构件复用者开发相应的用户界面和编写粘合代码均能起到一定的指导作用。

4.4 规约的自动化处理

为满足规约自动化处理的需要,本文给出的构件规约形式很容易转换为 XML 描述格式。XML 已经成为一种目前数据交换的唯一公共语言,很容易编程实现数据的表示和读取。图 9 是图 8 所示服务规约的 XML 格式表示。

```
〈? xml version="1.0" encoding="ISO-8859-1"?〉
〈ATOMIC_SERVICE_SECTION〉
 〈AS〉
  〈MS〉PlanQuery()〈/MS〉
  〈Pre〉显示系统管理页面并点击【计划审批】〈/Pre〉
  〈Post〉显示待审批的新用水计划〈/Post〉
〈Invarint〉":WaterPlanComp"存在 &RsvrInfoDB 打开〈/ Invarint 〉
 〈/AS〉
 〈AS〉
  〈MS〉PlanQuery();PlanChecked()〈/MS〉
  〈Pre〉显示新用水计划 & 点击【审批】链接〈/Pre〉
  〈Post〉审批记录存入数据库 ‖ "提交不成功"提示页面〈/Post〉
〈Invarint〉" :WaterPlanComp"存在 &RsvrInfoDB 打开〈/Invarint〉
 〈/AS〉
〈/ATOMIC_SERVICE_SECTION〉
〈AS_COMPOSITION_PATTERN〉
  〈PT〉[AS3]* ‖ [AS4]* 〈/PT〉
〈/AS_COMPOSITION_PATTERN〉
```

图 9 图 8 所示服务规约的 XML 格式表示

结束语 本文提出并制定面向服务的构件行为规约(SOCS)。面向服务的构件规约描述反映了构件的本质特征,它不仅清晰地表达构件自身的设计逻辑特征以及复用者角度的构件接口视图,而且提供了一种从服务角度考虑并解决 CBD 过程中相关问题的基本思路。

参 考 文 献

[1] Lüders F, Lau K-K, Ho S-M. Specification of Software Components[C] // Building Reliable Component-based Software Systems(Chapter 2). Artech House, 2002: 23-38

[2] OMG. CORBA Component v3.0 formal/02-06-65[OL]. http://www.omg.org/technology/documents /corba-spec-catalog.htm

[3] Finney K. Mathematical Notation in Formal Specification: Too Difficult for the Masses? [J]. IEEE Translation Software Engi-

neering,1996,22(2):158-159
[4] Matsumoto M, Futatsugi K. Highly Reliable Component-Based Software Development by Using Algebraic Behavioral Specification[C]//Proceedings of 3rd IEEE International Conference on Formal Engineering Methods,2000. IEEE Press,2000:35-43
[5] Eriksson H-E,Penker M. UML Toolkit [Z]. Publisher by John Wiley & Sons,Inc. 1997
[6] Küster J. Filipe:A logic-based formalization for component specification[J]. Journal of Object Technology,2002,1(3):231-248
[7] 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(4):721-732
[8] 周晓峰. 基于语义的软件构件匹配方法及在水利领域中应用的研究[D]. 南京:河海大学,2006
[9] Cervantes H, Hall R S. Automating Service Dependency Management in a Service-Oriented Component Model[C]// Proceedings of the 6th Workshop of the European Software Engineering Conference/Foundations of Software Engineering Component Based Software Engineering. September 2003:379-382
[10] Cervantes H,Hall R S. Autonomous Adaptation to Dynamic Availability Using a Service Oriented Component Model[C]// Proceedings of the 26th International Conference on Software Engineering. ICSE2004:614-623
[11] 孟凡超,初佃辉,战德臣,等. 基于服务规约匹配的业务构件获取[J]. 哈尔滨工业大学学报,2010,42(7):1112-1116

(上接第 147 页)

栈中所保存的路径驱动信息包括以下内容:

① 条件转移指令,包括指令地址和代码;

② 指令执行状态,包括 CPU 寄存器值和程序使用的内存单元内容。

执行控制模块根据指令分析的结果和路径管理模块保存的信息调度模拟调试环境执行相应的指令,向模拟调试环境提供待执行的指令信息和程序状态信息,触发模拟仿真环境执行指令。

分析结果生成模块综合记录和保存的程序信息生成分析结果报告,这里给出被分析程序的控制流图和汇编程序。

5 测试与分析

为了测试本文所提算法的有效性和实用性,实现了基于路径驱动的多路径二进制程序分析原型系统 D-CFG,建立了测试环境。硬件环境为 Dell 计算机系统,CPU:Intel Pentium Dual E2200 2.19GHz;内存:2 GB,硬盘:160 GB;OS 系统为 Windows XP SP3,D-CFG 由 C++语言实现。

为了说明算法的效果,以生成的控制流图作为比较的对象,测试和比较的内容包括生成控制流图中的基本块数和边数。测试结果与采用传统的动态执行方法(不包含路径驱动)的结果以及 IDA Pro 的执行结果进行对比。IDA Pro 是目前逆向分析领域较为成熟和流行的反汇编器,它支持不同类型处理器和不同格式的可执行文件,采用递归遍历反汇编策略对可执行文件进行反汇编处理,从而提取目标代码的控制流图,并且它还提供了可扩展接口。插件(plug-in)机制可以提供应用分析、动态调试、文件格式支持、处理器支持的插件扩展功能[8]。

选取 SPEC2000 的 CINT2000 程序集中的几个典型程序,将编译后的二进制程序代码作为测试集。令 b_{D_CFG} 表示 D-CFG 识别的基本块,b_{D_OLD} 表示传统动态执行识别的基本块,$cb_{D\text{-}OLD}=\frac{b_{D\text{-}CFG}\cap b_{D\text{-}OLD}}{b_{D\text{-}OLD}}\times 100\%$ 表示 D-CFG 识别的基本块对传统动态执行识别基本块的包含度,b_{IDA} 表示 IDA 识别的基本块,$cb_{IDA}=\frac{b_{D\text{-}CFG}\cap b_{IDA}}{b_{IDA}}\times 100\%$ 表示 D-CFG 识别的基本块对 IDA 识别基本块的包含度;$e_{D\text{-}CFG}$ 表示 D-CFG 识别的边数,$e_{D\text{-}OLD}$ 表示传统动态执行识别的边数,$ce_{D\text{-}OLD}=\frac{e_{D\text{-}CFG}\cap e_{D\text{-}OLD}}{e_{D\text{-}OLD}}\times 100\%$ 表示 D-CFG 识别的边对传统动态执行识别边的包含度,e_{IDA} 表示 IDA 识别的边数,$ce_{IDA}=\frac{e_{D\text{-}CFG}\cap e_{IDA}}{e_{IDA}}\times 100\%$ 表示 D-CFG 识别的边对 IDA 识别边的包含度。表 1 给出 D-CFG、传统动态执行和 IDA 的测试结果。

表 1 测试结果

程序名称	$b_{D\text{-}CFG}$	$b_{D\text{-}OLD}$	cb_{D_OLD}	b_{IDA}	cb_{IDA}	$e_{D\text{-}CFG}$	$e_{D\text{-}OLD}$	ce_{D_OLD}	e_{IDA}	ce_{IDA}
mcf	1672	1367	100%	1583	100%	2061	1526	100%	1738	100%
bzip	1489	1245	100%	1435	100%	2166	1629	100%	1840	100%
gzip	1580	1250	100%	1557	100%	2283	1835	100%	2007	100%
twolf	6453	5800	100%	5128	100%	7528	6538	100%	7106	100%
parser	6328	5783	100%	6015	100%	7513	5001	100%	5968	100%
crafty	7137	6779	100%	6538	100%	12584	8932	100%	11175	100%

从测试结果可看出,D-CFG 采用路径驱动的多路径方法较好地解决了动态执行的覆盖率问题,识别路径的覆盖率远远高于传统的动态分析方法,也优于目前广泛应用的 IDA。

结束语 动态多路径分析方法的目标是提高分析的路径覆盖率。本文提出的基于自动路径驱动的多路径分析算法,通过在转移分支指令处修改程序计数器 PC 值,驱动程序执行不同路径,可以有效提高分析路径的覆盖率。但是,转移指令有多种情况,目前的方法还不能完全发掘所有的程序路径,完善间接调用和变址转移的处理是我们下一步的主要任务。

参 考 文 献

[1] Bai Li-li, Pang Jian-min, Zhang Ping. Analysing Indirect Table Based on Critical Semantic Subtree[C]// 2010 International Conference on Compuer Design and Application, ICCDA2010. Volume 1,2010:9-13
[2] Linn C,Debray S. Obfuscation of executable code to improve resistance to static disassembly[C]// Proceedings of the 10th ACM Conference Oll Computer and Communications Sacufity. Washington Dc,USA,2003:290-299
[3] Song D, Brumley D, Yin H, et al. BitBlaze:a new approach to computer security via binary analysis[C]// Proceedings of the 4th International Conference on Information Systems Security. Berlin:Springer,2008:1-25
[4] 王祥根,司端锋,冯登国,等. 基于代码覆盖的恶意代码多路径分析方法[J]. 电子学报,2009,37(4):701-705
[5] Cifuentes C. Reverse Compilation Techniques[D]. Queensland: Queensland University of Technology,1994
[6] 胡刚,张平,李清宝. 基于静态模拟的二进制控制流恢复算法[J]. 计算机工程,2011,37:276-281
[7] 胡刚. 固件程序代码逆向分析关键技术研究[D]. 郑州:解放军信息工程大学信息工程学院,2010
[8] Micallef S. IDA Plug-In Writing In C/C++ [EB/OL]. http://www.binarypool.com/idapluginwriting/,2009