第35卷 第21期 计 算 机 工 程 2009年11月
Vol.35 No.21 Computer Engineering November 2009
·软件技术与数据库· 文章编号：1000—3428(2009)21—0060—02 文献标识码：A 中图分类号：TP311

基于代数规约的 Web 服务测试

余 波[1,2]，孔 良[2]，彭 琛[1]
(1. 中南林业科技大学涉外学院，长沙 410004；2. 国防科学技术大学计算机学院，长沙 410073)

摘 要：针对自动测试 Web 服务，提出基于代数规约测试 Web 服务的方法，包括描述 Web 服务的代数规约语言 ASOWS。基于 Web 服务的代数规约，采用方法覆盖准则以及等式覆盖准则自动生成测试用例，在此基础上，设计并实现一个原型工具。结果表明该方法能够自动测试部署在 Web 应用服务器上的 Web 服务。

关键词：Web 服务；代数规约；方法覆盖准则；等式覆盖准则

Web Service Test Based on Algebraic Specification

YU Bo[1,2], KONG Liang[2], PENG Chen[1]
(1. Swan College, Central South University of Forestry and Technology, Changsha 410004;
2. Department of Computer, National University of Defense Technology, Changsha 410073)

【Abstract】 For testing Web services automatically, the method testing Web service based on algebraic specification is proposed. The algebraic specification language ASOWS for describing Web service is represented. The test cases are automatically generated based on algebraic specification of Web service according to the method coverage criterion and the equation coverage criterion. The prototype tool for testing Web service is implemented. Experimental result shows that the method can be applied to test Web service automatically deployed on Web application server.

【Key words】 Web service; algebraic specification; method coverage criterion; equation coverage criterion

1 概述

Web 服务已经成为面向服务体系结构的主流实现技术，随着其应用越来越广泛，保证 Web 服务质量的测试也受到越来越多的关注。由于 Web 服务具有只对外公开其调用接口、隐藏其内部实现、在调用时动态绑定等特点，因此其测试不同于以往的集成测试。

基于形式规约的测试方法通常采用随机策略生成测试数据，具有测试自动化的特点[1]。基于代数规约的测试技术已经成功应用于过程语言表示的抽象数据类型[2]、面向对象程序的类[3]以及 EJB 组件[4-5]等程序的测试。实践表明，应用代数规约语言描述软件系统时，其规约具有规约简短、精确且独立于实现语言的特点，能够在不暴露对象实现细节的前提下描述软件实体的可观察行为，更重要的是，它能够应用代数规约中的公理等式充当判定测试是否通过的测试神谕(Test Oracle)，测试自动化程度较高。

基于代数规约的测试方法也可用于 Web 服务的自动化测试。但是要求描述 Web 服务的代数规约是良构的且其结构与 Web 服务的程序结构相匹配[5]，即规约定义的分类(Sort)的名称必须与 Web 服务名相同；签名部分的操作名、输入和输出类型与 Web 服务的接口一一对应。

2 Web 服务的代数规约语言

一个软件系统的代数规约除了定义数据和操作的签名(signature)外，另一个重要组成部分是刻画软件系统行为属性的公理集合。

定义 1 签名 $\Sigma =< S, F >$ 由 2 个部分组成：

(1)集合 S 的元素称为分类或类子；

(2) $f \in F$，且 $f : s_1 \times \cdots s_n \to s, s_1, \cdots, s_n, s \in S$。

定义 2 在代数规约 $SP =< \Sigma, E >$ 中，Σ 表示软件实体的形式规约的签名；E 表示软件实体的可观察行为的公理等式集合。

描述 Web 服务的代数规约语言 ASOWS 以描述组件的代数规约语言 CASOCC[5]为基础。两者的主要区别为：CASOCC 将操作分为创建子(creator)、构造子(constructor)、转换子(transformer)和观察子(observer)4 类；而 ASOWS 将创建子并入构造子。其余部分的语法和语义与 CASOCC[4-5]的相同。ASOWS 描述 Web 服务的基本单位是规约单元，以关键字 Spec 开始，end 结束，observable 后附带“F”表示该代数规约不可观察，“T”表示可观察；import 后附带引入的类子或其他分类；operations 部分依次是构造子、转换子和观察子的定义；Axioms 部分描述公理等式；Vars 部分描述公理中出现的变量。类子即基本类型的分类。预定义的类子有：byte, char, short, int, long, float, double, String 和 Boolean 及上述类型的数组。ASOWS 的 EBNF 范式如下：

```
<Spec Unit> ::=
Spec < Sort Name > Observable: <Boolean>
  [Import: <Import Sort List>]
  Operations: <Operation List>
  [Var:   <Variable Declaration List>]
```

基金项目：湖南省自然科学基金资助项目(06JJ5116)；湖南省普通高等学校教学改革研究项目(2008-248)

作者简介：余 波(1969－)，男，高级工程师、博士研究生，主研方向：软件测试，Web service 技术；孔 良，工程师、硕士研究生；彭 琛，助教、硕士研究生

收稿日期：2009-07-15 **E-mail：**hnaxdsjk@163.com

[Axioms: <List of axioms>]
End

签名定义的分类称为主分类。如果 x 是主分类，操作 f 作用于参数 x 和 y，则可记为 $x.f(y)$。

定义 3[5] 代数规约 $SP=<\Sigma,E>$，签名 $\Sigma=<S,F>$，$s_1,s_2,\cdots,s_n$，O 是 S 的引入分类，且 O 是可观察分类，$s_i\neq S$，$0\le i\le n$，f 是 S 中的操作，$0\le k\le n$；如果 $f:s_0\times\cdots s_k\to S$，则称 f 为构造子；如果 $f:S,s_0\times\cdots s_k\to S$，则称 f 为转换子；如果 $f:S,s_0\times\cdots s_k\to O$，则称 f 为观察子。

ASOWS 的公理的 EBNF 范式如下：

<Axiom> ::= <Label> : <Equation>[, if <Condition>]
<Label>::= <Number> | <Identifier>
<Equation> ::= <Term> = <Term>
<Condition> ::= <Term of Boolean type>
| <Equation> | <Term> <Relation Operator> <Term>
| <Condition> <Logic Connective> <Condition>

一个项由 Var 部分声明的变量、Operations 部分声明或引入分类的操作或常数构成。一个等号连接 2 个项，构成一个公理等式。条件可以是布尔型的项、等式以及逻辑关系式。一个公理等式包含标号、等式和可选条件 3 个部分。等式中每个包含操作的项由构造子或类型为主分类的变量开始，连接转换子，最后以观察子结束。

堆栈 Web 服务的代数规约如下：

```
Spec StackService
    observable F;
    import int,String;
    operations
      creator
          stack:String->StackService;
      transformer
          push:StackService,int->StackService;
          pop:StackService->StackService;
      observer
          getId:StackService->String;
          top:StackService->int;
          height:StackService->int;
    vars
      S:StackService;     n:int; x:String;
    axioms
     1:stack(x).getId() = x;
     2:findByPrimaryKey(x).getId() = x;
     3:stack(x).height() = 0;
     4:S.push(n) = S; if S.height() = 10;
     5:S.pop() = S; if S.height() = 0;
     6:S.push(n).pop() = S; if S.height() < 10;
     7:S.push(n).top() = n; if S.height() < 10;
     8:S.push(n).height() = S.height()+1; if S.height() < 10;
     9:S.pop().height() = S.height()-1; if S.height() >0;
end
```

代数测试的基本思想为：以常数替代公理等式的项的变量时，该项即为基础项。由 1 条公理可得到 2 个相等的基础项。包含操作的项又能看成操作或方法的调用序列，据此可构造测试用例[5]。因此，测试用例可看作是一个三元组<*T*1, *T*2, *Cond*>，其中，*T*1 和 *T*2 是基础项；*Cond* 是 Boolean 分类的可选条件。当 *Cond* 为真时，*T*1 和 *T*2 的值应该相等。如果不等，则被测试程序中存在错误。

3 代数规约的完备性和测试覆盖准则

从逻辑角度而言，完备性有 3 种表示形式：(1)弱完备性即公理集中的所有的永真式均是可证的。(2)演绎完备性在每个语义蕴涵在证明系统中可推导时成立。(3)强完备性在每个语法理论是某个“最小”模型的语义理论时成立。选择完备的公理集合设计测试是基于代数规约的测试提高测试覆盖度的基础。虽然判定公理集合的完备性是困难的，但是引入项重写规则系统则可以解决这个难题。而且在实际测试时，还可从公理集中选择有针对性的部分公理安排测试。

测试覆盖准则是选择测试数据和评价测试效果的基础。由于结构测试的语句和路径等覆盖准则不能够直接应用于基于规约的测试用例的设计，因此需要定义新的测试覆盖准则。

定义 4 方法覆盖准则(Method Coverage Criterion, MCC)选择足够的测试用例，使得规约定义中的每个方法至少被执行 1 次。

方法覆盖准则是一个覆盖程度较弱的准则。

定义 5 法组合覆盖准则(Method Combination Coverage Criterion, MCCC)选择足够的测试用例，使得规约中任何一个构造子及转换子与观察子的组合至少执行 1 次。

定义 6 等式覆盖准则选择足够的测试用例，对于 $SP=<\Sigma,E>$ 中任何一个等式 $e,e\in E$，使得 e 在其条件为真时至少执行 1 次。

覆盖率=至少执行一次的项数/总项数。当测试用例集 T 满足方法组合覆盖准则时，覆盖率等于 100%。此时规约中的方法及其组合也必须在公理集中至少出现 1 次。根据定义：T 也满足方法覆盖准则，反之不然。与前两者不同，等式覆盖侧重考察测试用例所覆盖的公理等式。等式覆盖准则可以与前 2 个覆盖准则组合使用。公理分析算法检查签名定义中的全部操作是否在公理中出现。显然，基于代数规约充分测试 Web 服务，要求该公理集合是完备的且根据方法组合覆盖准则并选择等式覆盖准则生成测试用例。

算法 公理分析算法
输入 method set *M*, Axiom set *A*; //each equation is an axiom;
输出 the methods which are the same time in *M* and *A*;

```
// A ⊕ B = {x | !(x ∈ A ∧ x ∈ B)}；
F=M; G={};
while(A!=Null){ get an equation e; A=A-e;
   split left term L and right term R from e;
   while(L!={}^R!={}){
        get method l from L; L=L={l};
        get method r from R; R=R-{r};
         G=G∪{l}∪{r};
         F=F⊕  ({l}∪{r});}}
return (G ⊕ M)  ∪F; //end of the algorithm 1;
```

4 原型工具体系结构及实现

原型工具 WSTAS 可以测试部署在 Web 应用服务器上的 Web 服务。它可根据方法、方法组合和等式覆盖准则设计被测试 Web 服务的测试用例，前提是签名定义中的全部方法及其组合要在公理集合出现。它主要由规约处理模块、测试执行模块和测试评估模块 3 个部分组成。

(1)规约处理模块：规约编辑器提供创建、修改代数规约的编辑环境；规约分析器以 Web 服务的规约输入，对其进行语法分析，如果公理集合存在语法错误或不满足方法覆盖准则，则报告之；测试用例产生器在公理集合中选择部分或全部公理随机生成测试用例及相应的 test Oracle[5]。

（下转第 64 页）