一种 Web 服务的测试数据自动生成方法

姜 瑛[1),2)] 辛国茂[1)] 单锦辉[1)] 张 路[1)] 谢 冰[1)] 杨芙清[1)]

[1)] (北京大学信息科学技术学院软件研究所 北京 100871)

[2)] (昆明理工大学信息工程与自动化学院 昆明 650093)

摘 要 软件测试是保证 Web 服务质量的重要技术手段. 测试数据生成是 Web 服务测试的重要内容. 测试数据的质量将直接影响 Web 服务测试的效率和成本. 文章基于合约式设计的 Web 服务测试技术, 提出一种 Web 服务的测试数据自动生成方法. 首先根据 WSDL 文档采用随机法自动生成初始测试数据, 然后使用合约变异技术进行测试数据的选择, 据此可以生成一组达到一定合约变异充分度的有效测试数据, 从而提高 Web 服务的测试质量和效率. 最后实现了一个 Web 服务的测试数据自动生成工具原型, 并通过实验验证了方法的有效性.

关键词 Web 服务; 软件测试; 测试数据生成; 合约; 变异测试

中图法分类号 TP311

A Method of Automated Test Data Generation for Web Service

JIANG Ying[1),2)] XIN Guo Mao[1)] SHAN Jin Hui[1)] ZHANG Lu[1)] XIE Bing[1)] YANG Fu Qing[1)]

[1)] (*Institute of Software, School of Electronics Engineering and Computer Science, Peking University, Beijing* 100871)

[2)] (*Faculty of Information Engineering and Automation, Kunming University of Science and Technology, Kunming* 650093)

Abstract Software testing is one of the most important techniques used to assure the quality of Web Services at present. Test data generation is an important topic in Web Services testing. The quality of test data will influence the efficiency and cost during Web Services testing. Based on the design-by-contract testing technology for Web Services, this paper presents a method of automated test data generation for Web Services. Firstly, according to the description information and contracts in WSDL document of Web Services, the initial test data are generated automatically by random method. Then the test data are selected by means of contract mutation testing. The algorithms about initial test data generation and test data selection are presented. In order to improve the quality and efficiency of Web Services testing, this method can generate a test suite meeting a certain contract mutation score. Finally, a prototype has been developed on the Microsoft .NET platform. For the same Web Service with different contracts, the experiments are carried out on this prototype. The contract mutation score of generated test data is compared with the statement coverage score and condition coverage score. The results have shown that the method is effective in automated test data generation for Web Services.

Keywords Web service; software testing; test data generation; contract; mutation testing

收稿日期: 2004-12-05; 修改稿收到日期: 2005-01-25. 本课题得到国家自然科学基金(60373003)、国家"九七三"重点基础研究发展规划项目基金(2002CB312003)和国家"八六三"高技术研究发展计划项目基金(2004AA112070)资助. 姜 瑛, 女, 1974 年生, 博士研究生, 主要研究方向为软件工程、软件测试、软件构件技术. E-mail: jiangy@sei.pku.edu.cn. 辛国茂, 男, 1981 年生, 硕士研究生, 主要研究方向为软件测试、软件构件技术. 单锦辉, 男, 1970 年生, 博士, 主要研究方向为软件测试、构件技术、形式化方法. 张 路, 男, 1973 年生, 博士, 副教授, 主要研究方向为软件工程、程序理解、配置管理、人工智能. 谢 冰, 男, 1970 年生, 博士, 副教授, 主要研究方向为软件工程、形式化方法、分布式系统. 杨芙清, 女, 1932 年生, 教授, 博士生导师, 中国科学院院士, 主要研究领域为系统软件、软件工程、软件工业化生成技术与系统.

1 引　言

Web 服务以 XML, SOAP, WSDL 和 UDDI 为核心, 具有良好的封装性和强大的集成能力, 支持开放、动态的互操作模式, 可极大地降低系统集成的复杂性和开销, 因此获得了产业界广泛的支持和学术界的重视[1].

为了保证 Web 服务的质量, 必须对其进行详尽的测试. Web 服务的测试内容主要包括: 测试 SOAP 消息; 测试 WSDL 文档; 测试面向服务的体系结构(Service-Oriented Architecture)的发布、发现及绑定能力; 模拟 Web 服务的提供者和使用者等. 此外, 还包括同步和异步测试、SOAP 中介(intermediaries)测试、服务级别协议(Service-Level Agreement)测试和服务质量(Quality of Services)测试以及 Web 服务的压力测试等等①②.

测试活动需要大量的测试数据. 测试数据的来源有三种: (1)基于功能. 从软件需求中获得; (2)基于结构. 从实现中获得; (3)基于错误. 从开发过程中产生的典型和有用的错误中获得[2]. 基于功能的测试数据通常由规约产生[3], 基于结构的测试数据可从程序语句的路径来生成[4], 文献[5,6]则基于消息交互的错误来获得测试数据. 据统计, 在所有的软件测试的开销中, 约 40%花费在设计测试用例(包括测试数据和期望结果)上, 约 50%花费在编写和编译测试脚本上, 另外约 10%花费在测试脚本的执行和配置管理上[7]. 当前, Web 服务测试主要依靠测试人员手工设计测试数据, 这种方法费时、费力, 且带有一定的盲目性和倾向性. 提高测试数据生成的自动化程度可以有效地减轻测试人员的劳动强度, 提高软件测试的质量. 此外, 测试数据的有效性将直接影响 Web 服务的测试效率和测试成本, 如何为 Web 服务自动生成有效的测试数据成为亟需解决的问题.

我们在以前的工作中, 针对 Web 服务使用中遇到的问题, 提出了一种基于合约式设计的测试技术[8]. 本文在上述工作的基础上, 提出一种 Web 服务的测试数据自动生成方法, 解决其中测试数据只能手工生成的问题. 由于在通常情况下, 用户无法获得 Web 服务的设计和实现细节, 不能基于其结构生成测试数据. 因此, 我们基于 Web 服务的功能说明, 即其 WSDL 文档中的方法描述及合约, 采用随机法来生成初始的测试数据. 本文首次将合约变异应用于 Web 服务测试数据的选择, 与贪心算法相结合, 对初始测试数据进行两次选择, 最终产生一组达到一定合约变异充分度的测试数据, 并通过对实际的 Web 服务进行多组实验, 对测试数据在选择前后的语句覆盖率和分支覆盖率以及这两种覆盖率与合约变异充分度进行比较, 验证本文方法的有效性. 选择后的测试数据可用来进行 Web 服务的回归测试, 或供其它用户复用, 从而减少测试成本.

本文第 2 节介绍基于合约式设计的 Web 服务测试技术, 它是测试数据产生的基础; 第 3 节简要介绍变异测试的基本原理; 第 4 节论述 Web 服务初始测试数据自动生成的方法; 第 5 节给出在初始测试数据中进行选择的算法以及对选择后测试数据的度量; 第 6 节给出使用本文方法进行的实验, 并对实验结果进行分析和评价; 第 7 节介绍相关工作, 并与我们的工作进行比较; 最后在第 8 节对本文工作进行了总结和展望.

2 基于合约式设计的 Web 服务测试

测试 Web 服务很困难, 其原因在于: 服务使用者需要选择和调用 Web 服务, 在使用中还涉及到 UDDI, SOAP, WSDL 等标准协议; Web 服务是包含大量运行态行为的分布式应用, 分布式系统具有事件难以重现、竞争和死锁等特性; 如果出于知识产权保护等原因, 服务提供者不愿意暴露过多有关 Web 服务的设计或实现细节, 那么服务使用者通常只能获得 Web 服务的接口信息即 WSDL 文档, 从而只能根据接口信息进行黑盒测试.

通常, 服务使用者在使用 Web 服务的过程中会遇到如下的问题: 一是无法完全了解 Web 服务的正确使用方式, 从而在测试时不能对发现的错误进行准确的定位, 即服务使用者无法分辨究竟是 Web 服务本身的错误, 还是使用 Web 服务的方法有误; 二是很难测试 Web 服务是否符合使用需求. 为了解决上述问题, 我们提出了一种基于合约式设计的测试技术[8]. 合约表明了服务提供者和服务使用者相互的义务和利益, 可用于区分软件失效时的责任: 如果前置条件被违反, 则应该在使用者处寻找错误; 如果

① Bloomberg J.. Testing Web services today and tomorrow. The Rational Edge E-zine for the Rational Community 2002, http://www.therationaledge.com/content/oct_02/PDF/WebTesting_TheRationalEdge_Oct02.pdf

② Davidson N.. Web services testing. The Red-gate software technical papers 2002, http://www.red-gate.com/dotnet/more/web_services_testing.htm

后置条件被违反，责任在提供者. 服务提供者和服务使用者均可在 WSDL 文档中定制合约，对 Web 服务中的方法增加前置条件和后置条件描述，即对 Web 服务的功能进行更进一步的说明，以提高 Web 服务的易测试性. 通过对 WSDL 语法的扩展，增强了对 Web 服务功能的语义描述，有助于在 Web 服务测试中区分服务提供者和使用者的责任.

基于合约式设计的思想，我们提出并实现了一个 Web 服务的测试技术框架，可以添加服务提供者和使用者的合约，根据测试数据生成 XML 格式的测试脚本，在测试执行时进行合约检查，返回测试结果并提交测试报告.

3 变异测试

变异测试的思想最早由 DeMillo，Lipton 和 Sayward 在 1978 年提出，它基于程序员能力假设和组合效应假设，是一种排错性测试技术[9]. 变异测试的基本原理是使用变异算子对原程序进行作用，产生大量的变异体，根据已有的测试数据，运行变异体，比较变异体的运行结果和原程序的运行结果. 如果两者不同，就称该测试数据将该变异体杀死. 导致变异体不能被杀死的原因有两个：(1) 测试数据集还不够充分，通过扩充测试数据集便能将该变异体杀死；(2) 该变异体在功能上等价于原程序，此时无论怎样扩充测试数据集也不能将该变异体杀死. 后者的这类变异体被称为等价变异体. 变异测试为衡量测试数据集的充分性提供了客观的准则，这个准则就是变异充分度. 变异充分度是已杀死的变异体数目与所有已产生的非等价变异体数目的比值. 变异充分度值越大说明测试数据越有效，最理想的效果就是该值达到 1. 变异测试具有排错能力强、灵活性好、自动化程度高等优点，但是变异体的执行和存储需要大量的计算机资源，而且判断一个变异体是否等价于原程序是一个不可判定的问题.

变异测试起初是针对白盒测试提出来的，但它也能用于黑盒测试，例如函数调用接口的变异测试. 针对不同的变异对象，还出现了规约变异与合约变异等.

4 初始测试数据生成

为了提高 Web 服务测试的效率，降低其测试成本，需要为 Web 服务自动生成有效的测试数据. 第 2 节中描述的框架不能自动产生测试数据，只能依靠测试人员手工生成. 为了解决这个问题，我们在原有研究的基础上提出了一种自动生成测试数据的方法.

一个 Web 服务的测试应针对其对外发布的各方法，为了更好地刻画 Web 服务测试的相关信息，我们给出 Web 服务测试信息的形式定义.

定义 **1**. 一个 Web 服务的测试信息是一个四元组集合 $W=\{\langle O_i, C_i, TD_i, MC_i\rangle \mid 1\leq i\leq n\}$，其中 O_i 为该 Web 服务的第 i 个方法；$C_i=\langle C_{\text{pre}}, C_{\text{post}}\rangle$，是 O_i 的合约，由前置条件 C_{pre} 和后置条件 C_{post} 组成；TD_i 为 O_i 的有效测试数据；MC_i 为 TD_i 达到的变异充分度值.

前置条件 C_{pre} 定义了使用 Web 服务的方法应该满足的条件，它是对方法中输入参数的说明和限制. 我们先将前置条件转化为形如 $C_{\text{pre}}=\bigwedge_{1\leq j\leq m} C_{\text{pre}_j}$ 的合取范式(m 为合取项的个数)，每个合取项 C_{pre_j} 是仅对方法的单个参数进行约束的析取范式，其形式为 $C_{\text{pre}_j}=\bigvee_{1\leq k\leq l}(P_x\,\theta_k\,Value_k)$，其中 P_x 是方法中的参数名，它可能出现在多个 C_{pre_j} 中；$\theta_k\in\{<, >, ==, <=, >=, !=\}$；$Value_k$ 是 P_x 输入域中的常量.

P_x 的数据类型及前置条件确定了其取值区间. 我们可将影响参数取值的条件映射到输入域上，即针对方法 O_i 中的每个参数 P_x，根据其数据类型及前置条件，先采用等价类划分的方法将 P_x 的输入域划分为有效等价类和无效等价类，从不同等价类中随机生成初始测试数据集；然后以边界值分析方法作为其补充，选取正好等于、略大于或略小于等价类边界的值加入初始测试数据集.

算法 **1**. 初始测试数据生成.

输入：参数 P_x 的数据类型 $DataType$，含有 P_x 的 C_{pre}

输出：P_x 的初始测试数据集 $TestData_{P_x}$

```
DataRegion = InitialRegion(DataType)
For(j = 1 to m)
  DataRegion = DataRegion ∩ CalRange(C_pre_j)
TestData_Px = ∅
For(w = 1 to u)
  Begin
    DataRange_w = GetRange(DataRegion, w)
    TestData_Px = TestData_Px ∪
                  GenerateTestData(DataRange_w, Num)
  End
InvalidDataRegion = CalInvalidRegion(DataRegion)
For(w = 1 to v)
  Begin
```

$InvalidDataRange_w =$
$GetRange(InvalidDataRegion, w)$
$TestData_{P_x} = TestData_{P_x} \cup$
$GenerateTestData(InvalidDataRange_w, InvalidNum)$
End

在上述算法中, 使用的函数定义如下:

InitialRegion(*DataType*) 的返回值是计算机能够处理数据类型 *DataType* 的范围, 如 *float* 类型为(-3.402823E38, +3.402823E38). *DataRegion* 指有效等价类, 由一组不相交的输入区间构成, 即

$DataRegion = \bigcup_{1 \le w \le u} DataRange_w$, $\forall w_1 \neq w_2$,
$DataRange_{w_1} \cap DataRange_{w_2} = \varnothing$,

无效等价类的定义与此类似.

$CalRange(C_{pre_j})$ 计算 P_x 在前置条件 C_{pre_j} 下的取值区间.

GetRange(*DataRegion*, *w*) 返回 *DataRegion* 的第 *w* 个区间.

GenerateTestData(*DataRange*, *Num*) 采用随机法, 按照用户预先定义的数量 *Num*, 返回 *DataRange* 所代表的范围内符合均匀分布的测试数据集合, 其中包括边界值.

CalInvalidRegion(*DataRegion*) 返回无效等价类 *InvalidDataRegion*.

依次对 O_i 的所有参数执行上述算法, 可生成每个参数的测试数据. 例如, 有某 int 类型的参数 *i*, 其 $C_{pre}(i >= 0) \wedge (i <= 100)$, 用户定义在有效等价类中生成 6 个测试数据, 在无效等价类中生成 2 个测试数据. 执行算法 1 后, *i* 产生的测试数据是{-1, -587, 0, 1, 100, 99, 41, 7, 101, 513}. O_i 的初始测试数据 $TestData_{O_i}$ 应对其每个参数的所有生成的取值组合进行覆盖, 即每个参数生成的各个取值都至少出现一次. 假设 O_i 中某参数 P_x 的测试数据集为 $TestData_{P_x}$, 它有 $|TestData_{P_x}|$ 个初始测试数据; 有两个参数 P_{x_1}, P_{x_2} 的方法至少可以产生 $|TestData_{P_{x_1}}| \times |TestData_{P_{x_2}}|$ 个初始测试数据; 依此类推, 有 *n* 个参数的方法至少产生 $\prod_{1 \le y \le n} |TestData_{P_{x_y}}|$ 个初始测试数据. 可以看到, 这批测试数据的数量庞大, 我们希望进一步从这些测试数据中进行选择, 减少测试数据的个数.

5　测试数据选择

测试数据选择的目的是为了缩小测试集的范围, 同时保持测试数据的有效性, 以提高测试效率, 降低测试成本. 在传统软件测试中已经有这方面的研究, 包括划分策略(dividing strategies)、基于启发式的算法、最小覆盖等. 第 4 节中对 Web 服务方法的每个参数的各个取值进行覆盖, 生成的初始测试数据数量庞大, 其执行需要花费相当多的时间. 为了提高测试效率, 必须对这些数据进行选择, 在保证一定有效性的前提下尽量减少测试数据的数量. 我们用合约变异技术结合贪心算法, 从初始测试数据中进行选择.

5.1　合约变异

合约变异是一种变异测试. 在传统的变异测试中, 所产生的变异体数目和原程序的行数的平方成正比, 变异体的存储和执行需要大量的计算机资源. 与传统变异测试相比, 合约的规模小于程序, 其产生的变异体数目较小. 此外, 还可以通过定义有效的变异算子来减少合约变异体的数量, 从而降低变异测试的计算代价. 通过合约变异, 可以得到一组有效的测试数据, 它能够杀死大多数变异体. 变异体与原合约的差别则可以为 Web 服务的排错提供有用的信息.

Web 服务合约的来源是服务提供者的设计规约和服务使用者的实际需求, 它规定了使用 Web 服务所需的条件和 Web 服务能够完成的功能. 由于合约是功能规约的部分体现, 因此, 合约变异也可被视为规约变异的一种特例. 文献[10] 指出, 规约变异的目的不是为了发现规约中的错误, 而是为了发现由于规约被错误理解或实现所导致的程序中的错误. 因此, 对服务提供者的合约变异可视为提供者对设计规约的错误理解, 对服务使用者的合约变异则体现了使用者的不同需求. 我们通过对 Web 服务的合约进行变异, 从中选择测试数据. 下面先给出一个变异系统的定义.

定义 **2**.　Web 服务的变异系统为 $WS\_MAS = \langle WS, CM, MO \rangle$, 其中, *WS* 代表 Web 服务; *CM* 代表 Web 服务合约的变异算子; *MO* 则是变异预言(Mutation Oracle), 变异预言是区分原程序与变异体的程序或条件[11].

变异算子是变异系统的重要部分, 是产生变异体的依据. 与传统的程序变异类似, 文献[10] 通过对形式化的合约定义做微小的语法修改来产生变异后的合约, 但并未给出所采用的合约变异算子及变异预言. 我们借鉴了传统程序变异中的成熟变异算子, 针对 Web 服务的合约, 给出了合约变异算子的定义.

定义 3. 合约变异算子 $CM=\{PRP, VRP, COR, LCR, AOR\}$. 变异算子的具体描述如表 1 所示.

表 1 Web 服务的合约变异算子

算子名	英文描述	中文解释
PRP	Parameter Replacement	方法的参数替换
VRP	Value Replacement	常量替换
COR	Comparator Operator Replacement	比较运算符替换
LCR	Logical Connector Replacement	逻辑联结符替换
AOR	Arithmetic Operator Replacement	算术运算符替换

使用表 1 中的变异算子, 可对 Web 服务方法的合约进行变异, 产生合约变异体. 例如, 某方法有 3 个参数 i, j, k, i 有一个前置条件 $i>=0$, 使用变异算子 PRP, VRP, COR 及 LCR 对它进行变异, 产生的变异体集合为 $\{j>=0, k>=0, i>=-1, i>=1, i>0, i<0, i<=0, i==0, i!=0, !(i>=0)\}$. 由于合约语句比一般的程序少很多, 因此它生成的变异体个数和花费的时间都低于传统的程序变异. 合约变异体将与原 Web 服务组合形成 Web 服务的变异体.

在针对程序语句的变异测试中, 一般采用程序的运行结果作为区分变异体与原程序的条件. 由于我们不变异 Web 服务的内部实现, Web 服务的运行结果不会在变异前后发生改变, 因此将合约检查结果的一致与否作为判断变异体是否被杀死的条件.

我们用 C_i', C_{pre}' 和 C_{post}' 分别表示 C_i, C_{pre} 和 C_{post} 的变异体. $\forall t \in TestData_{O_i}$, 用 t 分别运行 O_i 及其变异体 O_i' 后, 检查合约 C_i 与 C_i' 中前置条件和后置条件的运行结果, 合约变异预言包含如下规则:

(1) 若 C_{pre} 与 C_{pre}' 的运行结果不一致, 则变异体 O_i' 被杀死;

(2) 若 C_{pre} 与 C_{pre}' 的运行结果一致, 则进一步判断:

(a) 若 C_{pre} 与 C_{pre}' 均被违反, 则变异体 O_i' 未被杀死;

(b) 若 C_{pre} 与 C_{pre}' 均未被违反, 则进一步判断:

(i) 若 C_{post} 与 C_{post}' 的运行结果一致, 则变异体 O_i' 未被杀死;

(ii) 若 C_{post} 与 C_{post}' 的运行结果不一致, 则变异体 O_i' 被杀死.

我们使用合约变异预言来判断变异体是否被杀死. 例如运行测试数据 t 后, 若 O_i 的 C_{pre} 和 C_{post} 均未被违反, 其变异体 O_i' 的 C_{pre}' 未被违反而 C_{post}' 被违反, 则认为 O_i' 被 t 杀死.

5.2 选择算法

第 4 节中产生了一批分布于有效等价类和无效等价类中的初始测试数据. 对 O_i 的初始测试数据 $TestData_{O_i}$ 而言, 由于 $TestData_{O_i}$ 根据 O_i 中参数的数据类型及前置条件生成, 无效等价类中的测试数据实际上违反参数的前置条件. 由于各参数的前置条件之间是合取关系, 如果 $TestData_{O_i}$ 中出现大于或等于 2 个无效等价类中的数据, 则经过一次合约变异后, O_i 的前置条件也必然被违反. 按照 5.1 节中给出的合约变异预言, 这样的测试数据无法杀死变异体. 因此, 我们可在运行变异体前先对 $TestData_{O_i}$ 进行一次选择, 将其中出现大于或等于 2 个无效等价类的测试数据删除. 在此将第一次选择后的测试数据集用 $TestData_{O_{is}}$ 表示, 假设其数据个数为 S.

接下来我们用第一次选择后的测试数据运行 Web 服务方法及其变异体, 根据合约变异预言来判断变异体是否被杀死, 采用贪心算法, 从初始测试数据集中选出能杀死最多变异体的测试数据.

算法 2. 测试数据选择.

输入: O_i 第一次选择后的测试数据集 $TestData_{O_{is}}$, O_i 的变异体集合 M_T

输出: 最终测试数据集 TD_i, 被 TD_i 杀死的 O_i 变异体集合 M_D

```
For (q= 1 to S)
  Begin
    T_q= GetTestData( TestData_Ois, q)
    DeadMutants_q= MutationTesting( T_q)
  End
TD_i= ∅
M_D= ∅
Do
  Begin
    DeadMutants_max= Max( M_T, DeadMutants)
    TD_i= TD_i∪ T_max
    M_D= M_D∪ DeadMutants_max
    M_T= M_T - DeadMutants_max
  End
While (DeadMutants_max≠ ∅)
```

在上述算法中, 使用的函数定义如下:

$GetTestData(TestData_{O_{is}}, q)$ 从待选测试数据集 $TestData_{O_{is}}$ 中取出第 q 个测试数据.

$MutationTesting(T_q)$ 使用测试数据 T_q 运行 O_i 及其所有变异体, 返回被 T_q 杀死的变异体集合 $DeadMutants_q$.

$Max(M_T, DeadMutants)$ 返回 $DeadMutants$ 中包含被杀死变异体个数最多的子集, $DeadMutants_{max}$ 中的变异体是由测试数据 T_{max} 杀死的. 其中 $DeadMutants=\{DeadMutants_1, DeadMutants_2, \cdots, DeadMutants_S\}$

且 $DeadMutants_T \subseteq M_T$.

算法 2 执行完毕后, 如果 $M_T \neq \varnothing$, 表示 M_T 中有未被杀死的变异体. 我们用合约变异充分度作为衡量 Web 服务方法的测试数据有效性的准则, 其定义如下.

定义 **4**.　合约变异充分度 $MC_i(O_i, TD_i) = |M_D|/(|M_T| - |M_E|)$, 其中 M_E 是 O_i 的等价变异体集合.

通过以上过程, 我们从初始测试数据中选出了一组测试数据, 它们能够杀死一定数量的变异体, 并可计算其合约变异充分度值.

6　原型与实验

为了验证本文所提出方法的有效性, 我们基于测试 Web 服务的原型 WSTT[8], 在 Microsoft. NET 平台上用 C ♯语言开发了一个测试数据生成工具(Web Services Test Data Generation Tool, WSTDGT)的原型. 图 1 是测试数据生成的流程图.

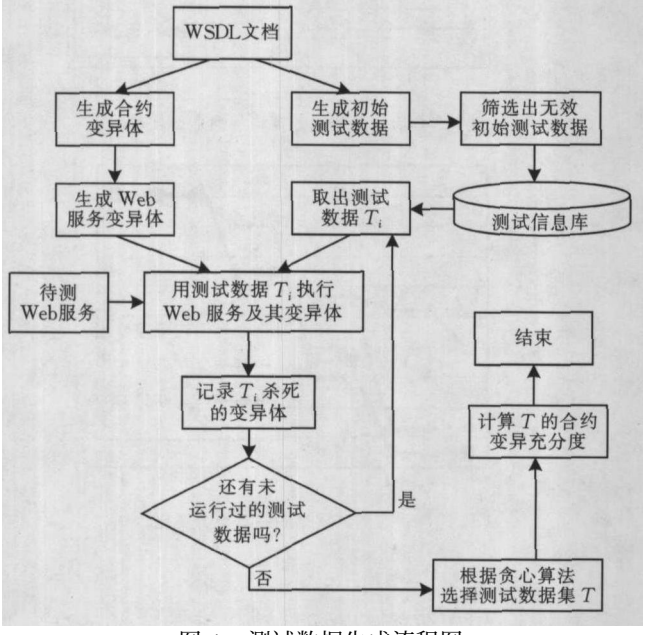


图 1　测试数据生成流程图

我们用一个实际的 Web 服务 TriTyp[12] 在 WSTDGT 上进行实验. 该 Web 服务只对外提供一个方法 public int[] TriTyp(int i, int j, int k), 其功能是根据三角形的边长判断三角形的类型, 返回值"1", "2", "3" 和"4"分别代表非等腰三角形、等腰非等边三角形、等边三角形以及错误的边长.

我们的实验环境是一台 2. 26GHz 主频、Pentium 4 CPU、512MB 内存的 PC 机, 操作系统是 Windows 2003 Server. 首先使用算法 1 生成初始测试数据, 然后对初始测试数据进行第一次选择, 再使用表 1 中的变异算子对 C_{pre} 和 C_{post} 进行变异, 最后执行算法 2 得到最终选中的测试数据.

在实验过程中, 我们使用了一对形如〈Num_l, Num_r〉的向量值, Num_l 和 Num_r 分别代表用户希望在参数的每个有效等价类和无效等价类中生成的测试数据个数. 我们对每组〈Num_l, Num_r〉输入都分别进行 5 次实验, 取其平均值作为最后的实验结果.

对于合约变异是否有效的问题, 当前还缺乏理论证明与实验分析. 本文希望通过实验, 借助语句覆盖率与分支覆盖率来说明其有效性. 因此, 我们分别用初始测试数据与最终测试数据运行 TriTyp, 计算它们各自的语句覆盖率和分支覆盖率, 并与合约变异充分度进行比较, 以此来说明本文中的合约变异充分度可以作为测试数据的度量准则.

针对 Web 服务 TriTyp, 我们设计了两组不同的合约, 分别进行实验.

6. 1　实验 **1**

我们为 TriTyp 书写的第 1 组合约如下:

$$C_{pre} = (i >= 0) \wedge (j >= 0) \wedge (k >= 0) \wedge (i <= 100) \wedge (j <= 100) \wedge (k <= 100),$$

$$C_{post} = (@return == 1) \| (@return == 2) \| (@return == 3) \| (@return == 4).$$

根据以上合约, WSTDGT 共生成 95 个合约变异体, 经我们人工判断, 其中 8 个为等价变异体. 例如, 一个变异体的 C'_{pre} 与 C_{pre} 相同, 其后置条件

$$C'_{post} = (@return == 1) \| (@return == 2) \| (@return == 3) \| (@return <= 4),$$

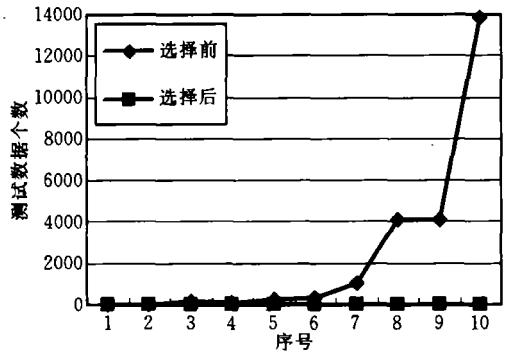
由于 TriTyp 只能返回"1", "2", "3", "4" 之一, 故 C'_{post} 等价于 C_{post}, 该合约变异体为等价变异体. 我们选择了 10 组〈Num_l, Num_r〉值, 使用本文方法得出实验结果如表 2 所示.

表 2 中, "初始数据生成时间"指执行算法 1、生成方法的初始测试数据并完成无效初始测试数据选择所花费的时间; "变异测试时间"指用第一次选择后的测试数据执行 Web 服务及其变异体, 进行第二次数据选择所花费的时间. 在测试充分度表项中, "选择前"指初始测试数据, "选择后"指使用本文方法后得到的最终测试数据.

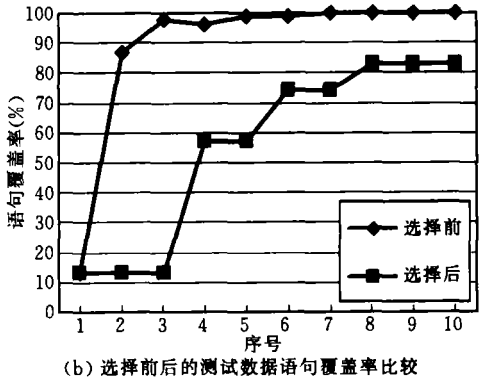
根据实验 1 的结果, 我们分别在图 2(a) ~图 2(d)中对选择前后的测试数据个数、语句覆盖率、分支覆盖率以及选择后的测试数据的测试充分度进行比较. 在这些图中, 横轴均代表不同的〈Num_l, Num_r〉, 用表 2 中第 1 列"序号"来表示.

表 2 实验 1 的结果

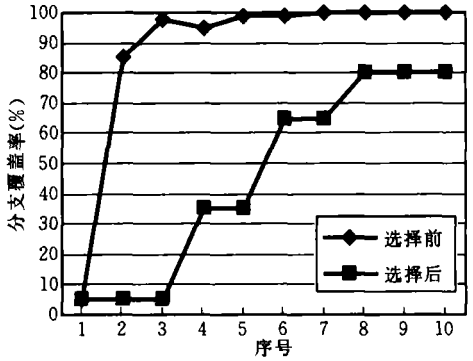
序号	〈Num_l, Num_r〉	初始测试数据个数	初始数据生成时间(ms)	最终测试数据个数	变异测试时间(ms)	选择前语句覆盖率(%)	选择前分支覆盖率(%)	选择后语句覆盖率(%)	选择后分支覆盖率(%)	合约变异充分度(%)
1	〈1, 0〉	1	147	1	20575	13	5	13	5	39
2	〈1, 1〉	27	75	7	14966	87	85	13	5	66
3	〈1, 2〉	125	241	7	15178	98	98	13	5	66
4	〈2, 1〉	64	175	8	17375	96	95	57	35	78
5	〈2, 2〉	216	441	8	18153	99	99	57	35	78
6	〈3, 2〉	343	734	9	24453	99	99	74	65	93
7	〈4, 3〉	1000	2109	9	38809	100	100	74	65	93
8	〈6, 5〉	4096	8669	10	99650	100	100	83	80	100
9	〈8, 4〉	4096	9772	10	185772	100	100	83	80	100
10	〈10, 7〉	13824	30166	10	381025	100	100	83	80	100



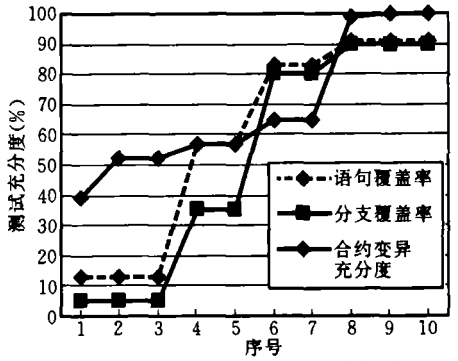
(a) 选择前后的测试数据个数比较



(b) 选择前后的测试数据语句覆盖率比较



(c) 选择前后的测试数据分支覆盖率比较



(d) 选择后的测试数据的测试充分度比较

图 2

6.2 实验 2

我们为 TriTyp 书写第 2 组合约如下:

$C_{pre}=(i>=0)\wedge(j>=0)\wedge(k>=0)\wedge$
$(i<=100)\wedge(j<=100)\wedge(k<=100)$,

$C_{post}=((@return==1)\|(@return==2)\|$
$(@return==3)\|(@return==4))\wedge$
$(!(@return==1)\|(i!=j))\wedge$
$(!(@return==1)\|(i!=k))\wedge$
$(!(@return==1)\|(j!=k))\wedge$
$(!(@return==1)\|(i>0))\wedge$
$(!(@return==1)\|(j>0))\wedge$
$(!(@return==1)\|(k>0))\wedge$
$(!(@return==1)\|((i+j)>k))\wedge$
$(!(@return==1)\|((i+k)>j))\wedge$
$(!(@return==1)\|((j+k)>i))$,

其中, 合约项$(!(@return==1)\|(i!=k))$实际上等价于$((@return==1)\rightarrow(i!=k))$. 根据以上合约, 共生成 284 个合约变异体, 其中有 86 个等价变异体. 我们同样选择了 10 组〈Num_l, Num_r〉值, 使用本文方法得出的实验结果如表 3 所示.

表 3　实验 2 的结果

序号	〈Num_l, Num_r〉	初始测试数据个数	初始数据生成时间(ms)	最终测试数据个数	变异测试时间(ms)	选择前语句覆盖率(%)	选择前分支覆盖率(%)	选择后语句覆盖率(%)	选择后分支覆盖率(%)	合约变异充分度(%)
1	〈1, 0〉	1	64	1	46531	13	5	13	5	39
2	〈1, 1〉	27	69	7	45475	87	85	13	5	52
3	〈1, 2〉	125	234	7	47175	99	99	13	5	52
4	〈2, 1〉	64	159	8	54056	96	95	57	35	57
5	〈2, 2〉	216	403	8	56169	97	97	57	35	57
6	〈3, 2〉	343	834	11	81883	99	99	83	80	65
7	〈4, 3〉	1000	2047	11	128178	100	100	83	80	65
8	〈6, 5〉	4096	8266	14	315663	100	100	91	90	99
9	〈8, 4〉	4096	9459	14	618447	100	100	91	90	100
10	〈10, 7〉	13824	22916	14	1214516	100	100	91	90	100

我们从实验 2 与实验 1 的结果中,各取其选择后的测试数据的语句覆盖率、分支覆盖率及合约变异充分度值,在图 3 中进行比较.

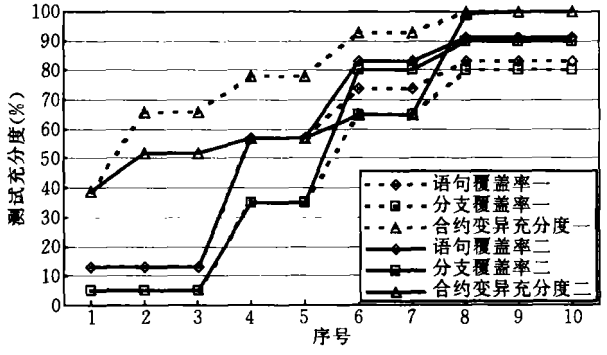


图 3　实验 1 与实验 2 中选择后的测试数据的 3 种测试充分度比较

实验 1 和实验 2 的结果说明本文给出的方法能够极大地缩小初始测试数据集. 可以看到,经过测试数据选择后,语句覆盖率和分支覆盖率会有一定的损失. 但最终选出的测试数据个数远远小于初始测试数据个数. 此外,图 3 显示,最终测试数据的合约变异充分度与其语句和分支覆盖率始终保持一致的增长趋势,可以从一定程度上说明合约变异充分度可以作为测试数据的度量准则,本文的方法是有效的.

实验 1 和实验 2 使用了不同的合约,从图 3 可以看出,用户对合约定义的精确程度将影响测试数据生成的质量. 由于实验 2 的后置条件比实验 1 更为详细和准确,因此针对相同的初始测试数据集(它们的前置条件完全相同),实验 2 的有效数据损失小于实验 1,其生成的最终测试数据的效果较好.

从表 2 和表 3 可以看出,初始测试数据生成和变异测试的时间均在可接受的范围内. 初始测试数据生成的时间取决于前置条件的定义和〈Num_l, Num_r〉值的选取;变异测试的时间取决于初始测试数据的规模和变异体的个数. 因此,用户对合约的定义将直接影响这两组时间.

7　相关工作

在当前 Web 服务测试的研究中,文献[13]对 Web 服务的 WSDL 文档进行了扩展,利用它生成测试计划,提出了一个支持测试执行和测试脚本管理的测试框架 Coyote,可对 Web 服务进行黑盒测试和回归测试. 为了提高 Web 服务的可靠性,文献[14]进一步提出在 UDDI Server 上通过 check in 和 checkout 机制增加验证特性以及执行分布式的测试机制. 对用户而言,通常需要集成多个 Web 服务来满足其需求,因此出现了对 Web 服务流(Web services flow)及服务动态组装的测试和验证,如文献[15],[16]和[17]. 文献[18]则是通过在网络层进行错误注入来评估 Web 服务的可靠性.

在 Web 服务的测试数据生成方面,文献[6]基于 Web 服务间传递的 SOAP 消息,采用了两种数据干扰(data perturbation)技术生成测试数据. 其中,数据值干扰(data value perturbation)根据数据类型用其边界值替换 SOAP 消息传递的值,从而产生测试数据. 交互干扰(interaction perturbation)包括 RPC 传递干扰和数据传递干扰,通过修改以上两类消息传递来产生测试数据. 该文的核心是修改 SOAP 消息的数据,如果测试人员无法获得相关数据,则无法使用此方法生成测试数据. 此外,该文并没有给出如何对生成的测试数据进一步精简的方法. 与之不同的是:(1)本文根据 Web 服务 WSDL 文档中的合约及相关描述,采用随机法生成初始测试数据;(2)本文使用合约变异技术对测试数据进行选择,可以极大地减少测试数据的个数. 实验表明,在选择的过程中会损失一些有效的测试数据. 可用

合约变异充分度值度量使用本文方法得出的测试数据. Web 服务方法的合约、测试数据及合约变异充分度值是相当重要的可复用信息,可供不同用户使用,有助于提高 Web 服务的测试和使用效率.

8 结束语

由于 Web 服务是一种包含大量运行态行为的分布式应用,因而不可能完全沿用传统软件测试技术对其进行测试. Web 服务的测试数据生成是 Web 服务测试的重要内容. 本文基于合约式设计的 Web 服务测试技术,提出了一种测试数据自动生成方法. 测试数据生成的基础是用户为 Web 服务制定的合约,本文的方法采用了随机法和合约变异相结合的技术,并开发了原型工具,通过实验说明该方法能够产生有效的测试数据.

目前, WSTDGT 原型仅能处理 int, float 及 double 类型的变量,文中所列举的合约变异算子还不够完全,如没有纳入用变量替换常量等变异算子. 实验结果表明,测试数据的选择过程并没有完全保持测试数据的有效性,这与书写的合约和初始测试数据的选择密切相关. 由于我们的方法目前只能处理 $P_x\ \theta_k\ Value_k$ 的情况,从一定程度上限制了合约的表达能力,从而将影响测试数据选择的有效性. 对于类似 $P_{x_1}\ \theta_k\ P_{x_2}$ 的情况,文献[19]给出了 P_{x_1} 与 P_{x_2} 取值范围的求解方法. 在今后的工作中,我们将针对这些问题做进一步的研究,希望更好地提高 Web 服务测试数据自动生成的质量和效率.

致 谢 本课题的实验工作得到北京大学信息科学技术学院软件研究所侯姗姗同学的协助,作者在此表示衷心感谢!

参 考 文 献

1 Yang Fu Qing, Mei Hong, Lu Jian, Jin Zhi. Some discussion on the development of software technology. Acta Electronica Sinica, 2002, 30(12A): 1901~1906(in Chinese)
(杨芙清,梅 宏,吕 建,金 芝. 浅论软件技术发展. 电子学报, 2002, 30(12A): 1901~1906)

2 Fabbri S. C. P. F., Maldonado J. C., Sugeta T., Masiero P. C.. Mutation testing applied to validate specifications based on statecharts. In: Proceedings of the 10th International Symposium on Software Reliability Engineering, Florida, USA, 1999, 210~219

3 Mandrioli D., Morasca S., Morzenti A.. Generating test cases for real time systems from logic specifications. ACM Transactions on Computer Systems, 1995, 13(4): 365~398

4 Shan Jin Hui, Wang Ji, Qi Zhi Chang. Survey on path wise automatic generation of test data. Acta Electronica Sinica, 2004, 32(1): 109~113(in Chinese)
(单锦辉,王 戟,齐治昌. 面向路径的测试数据自动生成方法述评. 电子学报, 2004, 32(1): 109~113)

5 Lee S. C., Offutt J.. Generating test cases for XML based Web component interactions using mutation analysis. In: Proceedings of the 12th International Symposium on Software Reliability Engineering, Hong Kong, China, 2001, 200~209

6 Offutt J., Xu W.. Generating test cases for web services using data perturbation. ACM SIGSOFT Software Engineering Notes, 2004, 29(5): 1~10

7 Tracey N. J.. A search based automated test data generation framework for safety critical software[Ph. D. dissertation]. Department of Computer Science, University of York, 2000

8 Jiang Ying, Xin Guo Mao, Shan Jin Hui, Xie Bing. Research on a testing technology based on design by contract. Journal of Software, 2004, 15(Supplement): 130~137(in Chinese)
(姜 瑛,辛国茂,单锦辉,谢 冰. 一种基于合约式设计的测试技术研究. 软件学报, 2004, 15(增刊): 130~137)

9 Zhu Hong, Jin Ling Zi. Software Quality Assurance and Testing. Beijing: Science Press, 1997(in Chinese)
(朱 鸿,金凌紫. 软件质量保障与测试. 北京: 科学出版社, 1997)

10 Aichernig B. K.. Mutation testing in the refinement calculus. Formal Aspects of Computing, 2003, 15(2~3): 280~295

11 Kovács G., Pap Z., Viet D. L., Wu Hen Chang A., Csopaki G.. Applying mutation analysis to SDL specifications. In: Proceedings of Specification and Description Language 2003, Stuttgart, Germany, LNCS 2708, 2003, 269~284

12 Sy N. T., Deville Y.. Automatic test data generation for programs with integer and float variables. In: Proceedings of the 16th Annual International Conference on Automated Software Engineering (ASE 2001), San Diego, CA, USA, 2001, 13~21

13 Tsai W. T., Paul R., Song W., Cao Z.. Coyote: An XML based framework for web services testing. In: Proceedings of the 7th IEEE International Symposium on High Assurance Systems Engineering(HASE' 02), Tokyo, Japan, 2002, 173~174

14 Tsai W. T., Paul R., Cao Z., Yu L., Saimi A.. Verification of web services using an enhanced UDDI server. In: Proceedings of the 8th International Workshop on Object Oriented Real Time Dependable Systems, Guadalajara, Mexico, 2003, 131~138

15 Nakajima S.. Verification of web service flows with model checking techniques. In: Proceedings of the 1st International Symposium on Cyber Worlds(CW' 02), Tokyo, Japan, 2002, 378~385

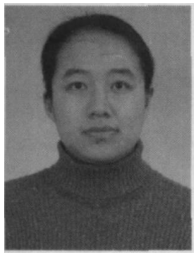
16 Foster H., Uchitel S., Magee J., Kramer J.. Model based verification of web service compositions. In: Proceedings of the

18th IEEE International Conference on Automated Software Engineering, Montreal, Quebec, Canada, 2003: 152~161

17 Tsai W.T., Chen Y., Paul R., Liao N., Huang H.. Cooperative and group testing in verification of dynamic composite Web services. In: Proceedings of the 28th Annual International Computer Software and Applications Conference, Hong Kong, China, 2004: 170~173

18 Looker N., Xu J.. Assessing the dependability of SOAP RPC-based Web services by fault injection. In: Proceedings of the 9th IEEE International Workshop on Object Oriented Real-Time Dependable Systems, Capri Island, Italy, 2003: 163~170

19 Offutt J., Jin Z., Pan J.. The dynamic domain reduction approach to test data generation. Software Practice and Experience, 1999, 29(2): 167~193



JIANG Ying, born in 1974, Ph. D. candidate. Her research interests include software engineering, software testing and software component technology.

XIN Guo Mao, born in 1981, master candidate. His research interests include software testing and software component technology.

SHAN Jin Hui, born in 1970, Ph. D.. His research interests include software testing, component technology and formal method.

ZHANG Lu, born in 1973, Ph. D., associate professor. His research interests include software engineering, program comprehension, configuration management and artificial intelligence.

XIE Bing, born in 1970, Ph. D., associate professor. His research interests include software engineering, formal method and distributed system.

YANG Fu Qing, born in 1932, professor, Ph. D. supervisor, member of Chinese Academy of Sciences. Her research interests include system software, software engineering, industrialized software manufacturing technology and system.

Background

Web Services have the potential to reduce the complexities and costs of software integration projects dramatically. It is difficult to test Web Services because they are distributed applications with numerous runtime behaviors. In order to test Web Services, a testing technology based on design by contract has been proposed early, which can improve the testability of Web Services and solve some testing problems better. Efficient test data can improve the quality of Web Services testing while decreasing its cost. Test data generation is an important task in Web Services testing. This paper aims at automated test data generation for Web Services based on the design by contract testing technology for Web Services. This research is supported by the National Basic Research Program of China (973 Program) under grant No. 2002CB312003, the National High Technology Research and Development Program (863 Program) under grant No. 2004AA112070, and the National Natural Science Foundation of China under grant No. 60373003.