计算机研究与发展
Journal of Computer Research and Development
DOI:10.7544/issn1000-1239.2017.20151045
54(3):609-622,2017

基于接口契约的有状态 Web 服务用例集生成

李 吟
(江苏自动化研究所 江苏连云港 222061)
(leein121999@126.com)

Test Suite Generating for Stateful Web Services Using Interface Contract

Li Yin
(*Jiangsu Automation Research Institute*, *Lianyungang*, *Jiangsu* 222061)

Abstract As Web services have the characteristics of only providing interface documents, complex technical specifications and run-time transient change, it is still a difficult problem to automatically generate test data effectively. At present, there is less current research on testing operation sequence for stateful Web services. Moreover, the existing approaches take insufficient account of service behavior information and the dependency between operations, and are lack of effective means of test automation, which may lead to high cost and short of specific for the test data. In this paper, a test case generation approach is proposed based on EFSM model operation interface contract. This approach constructs the operation model according to the standard WSDL document to describe the interaction relationship between operations and then add semantic annotation for them. Based on EFSM model, the paper proposes an automated operation sequences generation method and finally obtain the test suite using operation interface contract. The experiment shows that the proposed approach can generate reasonable test suite for stateful Web service efficiently, which can enhance the fault detection ability and improve the efficiency of test cases compared with the existed approaches.

Key words stateful Web services; operation sequence; operation interface contract; extended finite state machine (EFSM); semantic annotation

摘 要 Web 服务具有对外只提供接口文档、技术规范复杂和运行时态瞬时多变等特性,如何有效地自动化生成测试数据仍旧是个难题.目前,针对有状态 Web 服务操作序列的测试研究较少,现有的方法对服务的行为信息和操作之间的数据依赖关系考虑不足,且缺乏有效的测试自动化手段,进而导致测试代价较高以及生成的测试数据缺乏针对性.对此,提出一种结合有限状态机(EFSM)模型和操作接口契约的测试数据生成方法,该方法根据标准 WSDL 文档建立操作模型,形式化描述操作之间的交互关系,并对其进行语义标注,基于 EFSM 模型生成操作序列测试路径,随后采用操作接口契约关系获取测试路径中的测试数据.通过案例表明,这种有状态 Web 服务的用例生成方法能够高效地生成合理的测试数据,并在错误检测能力以及用例有效性方面优于现有的方法.

关键词 有状态 Web 服务;操作序列;操作接口契约;扩展有限状态机;语义标注

中图法分类号 TP311.5

Web 服务作为实现面向服务架构(service oriented architecture, SOA)系统的主要技术,根据标准的协议来描述、发布、编制、运行,并保证不同平台应用服务的互操作性.为保障 Web 服务的质量、

收稿日期:2015-12-09;**修回日期**:2016-06-14

提高 Web 服务的可信度,需要对 Web 服务进行系统、全面的测试.由于 Web 服务技术规范的复杂性、应用部署的网络分布性以及运行状态的多变性,传统的软件测试技术已难以适应 Web 服务的技术发展.尤其在服务测试方面,Web 服务自身提供的信息较少,不具备可视化图形界面,它的测试主要是基于接口进行设计和实现,与传统的人工干预产生测试数据的方法有着较大的区别,为了对其进行详尽的测试,需采用自动化的测试方法.

对于单个 Web 服务,测试人员可以通过标准 XML 规范文档中的数据描述信息执行黑盒测试.目前存在一些通过 WSDL(Web Services Description Language)文档中服务操作参数及类型的描述生成测试数据的研究,可是缺少对单个服务中存在的各类操作序列的深入研究,对 WSDL 文档进行分析的工作局限于数据类型,生成的测试用例无法覆盖服务中的测试路径,用例设计不完善,而有状态服务中的各个操作之间也可能包含较强的耦合联系.一个服务操作修改的全局性状态可能影响到另一个操作的执行.因此,一些开发过程中未充分考虑的特殊操作序列可能引发服务的质量缺陷,为服务的可靠运行带来风险.

本文提出了一种基于扩展的有限状态机(extended finite state machine, EFSM)以及接口契约模型的测试用例自动生成方法,用于生成有状态 Web 服务中存在的操作序列路径的测试用例集.将测试数据生成分为 2 部分:1)基于 WSDL 文档建立操作树模型,采用语义标注的方法扩展 WSDL 文档;2)根据状态划分构建操作调用序列的 EFSM 模型,并依据操作接口契约模型进行操作序列测试数据的选择和生成,以最优用例覆盖有状态 Web 服务存在的操作序列路径.

1 研究背景和相关工作

随着 SOA 这种技术的广泛应用,对 Web 服务的质量和正确性的要求越来越高,如何保障 Web 服务软件的质量和可靠性变成了当前软件工程领域迫切需要解决的难题.当前,Web 服务测试主要依靠测试人员手工设计测试数据,这种方法不仅效率不高,并且生成的数据带有一定的盲目性和倾向性,因此提高测试数据生成和执行的自动化程度成为亟需解决的问题[1-4].

Web 服务被分为无状态和有状态的 Web 服务 2 类[5-7].依据 Web 服务不同层次考虑的侧重点将 Web 服务测试划分为 3 个层次[8]:无状态 Web 服务中单个操作的测试、有状态 Web 服务中服务操作序列的测试以及基于业务流程编排的 Web 服务组合测试.有状态的 Web 服务含有多个操作,不同的操作序列组合往往导致 Web 服务进入不同的数据状态,因此,对于有状态的服务,测试操作序列比检验服务的状态更重要.

在无状态 Web 服务单个操作测试的研究方面,由于用户能够获取 Web 服务接口信息的 WSDL 描述文档,它包含了一些服务名称及参数类型等信息,从而可以采用基于规约的方法进行黑盒测试[9].Tsai 等人[10]提出从 4 个方面(输入输出依赖关系、调用序列、分层功能描述、并发序列说明)对 WSDL 文档进行扩展、增强其描述能力来提高 Web 服务的易测试性.Xu[11]和 Hanna 等人[12]通过解析 WSDL 文档将其转换成文件对象模型(document object model, DOM)树,进而建立形式化树的抽象模型,利用边界值分析和等价类划分等方法,随机为 Web 服务中的简单和复杂类型生成测试数据.随后,马春燕等人[13]在此基础上增加了复杂数据类型结构的描述,提出了一种更具可视化和可理解性的抽象模型,进一步细化数据之间的约束关系,更好地形式化描述测试用例的产生过程[13].同时,Sibilini 和 Mansour 等人在文献[14]探讨了通过变异测试技术实现单个 Web 服务的测试,并通过服务的响应来分析测试结果.在国内,北京大学姜瑛等人[15]基于合约式变异的方法应用于 Web 服务测试数据的选择及生成.

目前,对于有状态 Web 服务操作序列测试方面的研究主要通过描述被测系统的预期行为模型来生成测试数据.为了克服 WSDL 文件没有提供操作序列生成需要的 Web 服务行为信息(考虑输入输出前置后置条件)的限制,国内外研究者都开展了相应的研究,并取得了一定的进展.白晓颖等人[16]对 Web 服务的 WSDL 进行语义上的简单分析,采用 DOM 文档对象模型树,提出从测试数据生成、独立测试操作生成、操作流生成以及测试说明的工作流程生成测试用例的方法.在此基础上,Li 等人[17]引入 schema 特殊树模型,通过分析操作之间交换的数据消息进行操作序列的生成.而这种基于操作依赖的方法只能处理简单的数据类型,且缺乏具体的数据生成方法和扩展应用,具有一定的局限性.Bertolino 等人[18]则提出服务提供者应上载操作序列的协议

状态机,在此基础上,Heckel 等人[19]也提出服务提供者应同时上载 GT 规则(graph transformation rules)来描述 Web 服务行为信息.但这类方法缺乏相应的规范,以及增加了服务开发难度及工作量,很难得到实际的应用.此外,一些研究者基于加入了行为信息的语义 Web 服务标准,分别提出了将语义 WSDL 文档转换为事件序列图模型、IOPE 图模型、EFSM 有限状态机模型、SXM 模型等产生测试数据[20-25].但是这些模型大多数是针对带语义信息的 Web 服务说明,并且由于无法自动化生成数据,因此不能作为常规的测试手段.

从上述研究现状来看,文献[11-12]虽然提出了数据类型的抽象模型,但是在刻面约束较多的情况下无法有效地表示复杂数据类型的结构特点.文献[13-15]在此基础上为服务的输入数据定义形式化模型,并依此提取的数据模型生成测试数据,该方法针对 Web 服务中单个操作生成测试用例的方法研究相对比较完善,如果服务中包含操作之间的序列调用关系,则无法使用此方法生成合理的测试数据.Li 等人在文献[17]中仅从 Web 服务中操作的输入输出关系方面生成操作序列,对于其他操作之间的关联关系考虑不足,无法达到测试的充分有效.文献[18-19]提出的扩展方法不仅增加了服务开发的难度,并且生成的测试数据过度依赖服务注册时上传的 GT 规则,降低了软件的可测试性,容易导致测试用例生成的不充分.针对有状态的 Web 服务,Keum 等人[7]采用有限状态机的方法更好地描述了服务中存在的操作行为信息,可是模型构建及序列生成依赖于测试人员的经验,且自动化程度不高,降低了测试的效率.文献[20-25]描述了基于语义的测试路径的构建方法,但是考虑到标准的 WSDL 不提供相应的语义信息,此类方法很难得到广泛的应用支持.同时,上述方法均没有考虑到操作序列中测试数据之间的接口契约关系,以保证测试路径中的数据传递及测试路径中驱动数据的合理有效,且现有的操作序列的构建和测试数据的生成主要还是依赖于人工完成,过程比较繁琐耗时,存在明显的不足[26-27].

本文在此基础上,将建立基于 WSDL 文件的操作树模型,并对其进行语义的标注扩展,针对扩展后的 WSDL 语义文件进行自动化的有限状态机建模,从新的角度,采用基于接口契约的模型研究操作之间的数据关系并采用数据分区的方法,最终实现了有状态 Web 服务测试路径中的测试用例集的生成,提高了测试的自动化程度和保障了测试用例的合理有效.

2　模型及定义

2.1　基于 WSDL 描述的操作树模型

Web 服务采用 WSDL 文档对外进行服务描述[28-29],它是一个满足 W3C 系统规范的 XML 文档.针对有状态的 Web 服务,本文基于文献[17,30]提出的模型将单个 Web 服务的 WSDL 描述文档中操作部分形式化为一个逻辑上的树状结构,如图 1 所示.图 1 中根节点代表对应的 WSDL 文档;第 2 层的节点代表该 Web 服务包含的操作,分为操作的

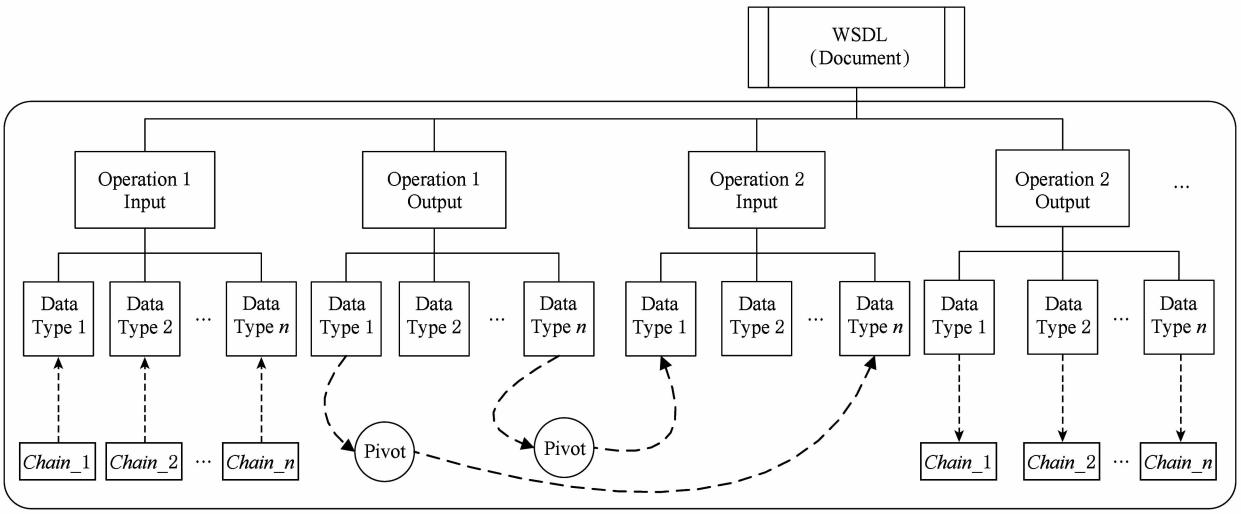

Fig. 1　Operation tree model

图 1　操作树模型

输入和操作的输出;第 3 层节点代表操作调用的输入输出数据;第 4 层节点代表输入输出参数之间的关联性.

定义 1. Web 服务操作树模型. 一个 WSDL 文档中的操作元素 $O=(Input(O), Output(O))$, 其中:

$Input(O)=\{M|M$ 是操作 O 的一个输入消息$\}$;

$Output(O)=\{M|M$ 是操作 O 的一个输出消息$\}$;

$M=\{P|P$ 是 WSDL 中 M 用来描述 O 的一部分$\}$;

$P=\{G|G$ 是和 P 相关的 XML Schema 定义$\}$.

根据定义 1,操作树模型的构建算法如下:

算法 1. 操作树模型 O 生成算法.

输入:WSDL 文档;

输出:WSDL 文档中操作元素形成的操作树模型 O.

```
CurLevelNode, NextLevelNode;
/* 定义当前层次及下一层次节点 */
int Level=0; /* 初始化层次数据,第 1 层为 0 */
Root=build_node(Schema, Tree, 0); /* 获取根节点 */
for 每个节点 Root.childNode()
/* 向下逐个遍历每一层 */
  CurLevelNode=NextLevelNode;
  /* 将下一层节点赋予当前层节点 */
  Level++; /* 当前层次遍历结束,层次加 1 */
  Transform_node(Node, Tree.Root);
  /* 从根节点向下遍历当前节点 */
end for /* 结束 for 循环 */
Transform_node(Node, Root){
  /* 遍历当前层次子节点函数 */
  if (Node.classtype=Element || Node.
    classtype=TYPE || Node.classtype=
    VALUE){
    /* 如果当前节点的类型是 element, type 或 value */
    Bulid(Node, Root);} /* 以此为根节点进行构建 */
  else{
    New_Root=Root;} /* 创建新的根节点数据 */
    for 每个节点 Node.children
      Transform_node(Node, New_Root);
      /* 迭代遍历当前节点 node */
  end for }
  end if
  Build_node(Node, Root, Level)
  /* 构造根节点数据函数 */
  {Child=new tree Node;
   Child.set_attributes(Node, level);
   Root.children+=child;
   Retun child;}
```

2.2 语义标注

目前广泛使用的 WSDL 文档没有提供 Web 服务的行为信息(操作的前置条件和结果),测试人员仅根据标准 WSDL 很难完成操作序列的测试. 为了增强 WSDL 语言的语义描述能力,本文基于文献[31-32]的方法,在现有的 Web 服务标准(即 WSDL 和 UDDI)基础上使用 WSDL-S 为标准 WSDL 文档添加语义信息,并采用 SWRL(semantic Web rule language)[33] 领域本体规则对 WSDL 文档进行语义标注.

使用 WSDL-S 为操作符添加语义标签包含 2 个要素:

1) "行为"标签描述操作符执行的行为. 使用行为标签可以指出本体中对应的类执行之后得出的结果,在 WSDL-S 文档中采用 effect 标签描述;

2) "约束"标签用于描述状态,可以描述该操作的进入条件,以确保操作的执行,在 WSDL-S 文档中采用 precondition 标签描述.

2.3 扩展有限状态机(EFSM)模型

有状态 Web 服务的测试类似于传统的面向对象软件的行为测试,行为状态包含控制状态(操作序列)以及数据状态(数据变量取值),可以依据文献[7,24]提出的 EFSM 模型中的状态转换来描述操作序列间的交互行为,以辅助测试路径生成. EFSM 的 Web 服务测试模型定义如下:

定义 2. EFSM 的 Web 服务形式化模型. 它是一个六元组$(Q, \Sigma, j, q_0, F, QC)$,其中:

1) Q 是一个有穷集合,叫作状态集,每一个状态表示交互序列中的历史记录或条件判断记录;

2) Σ 是一个有穷集合,叫作字母表,它是 $O\times I$ 的笛卡儿积集合;

3) I 是操作输入变量的集合,O 是操作输出变量的集合;

4) $\Phi: Q\times\Sigma\times QC\longrightarrow Q$ 是转移函数,Φ 表示下一个状态,即给定一个状态,根据输入输出参数条件,$EFSM$ 可以转移到另一个或几个状态;

5) $q_0\in Q$ 是起始状态;

6) $F\subseteq Q$ 是接收状态集,即用户与 Web 服务结束交互的状态集;

7) QC 是状态的条件集,其中条件是由状态和命题逻辑公式通过联接词组成的表达式.

根据 2.2 节的方法扩展后的 WSDL 文档包含了 Web 服务语义关系,因此采用语义规则语言 SWRL 来表示状态的条件集.

2.4　操作接口契约模型

本文根据侯可佳[34]、Li 等人[17]提出的模型进行改进,将单个操作的信息形式化为一个五元组的操作接口契约模型,用于捕获操作之间的依赖关系,定义如下:

定义 3. 操作接口契约模型. $Operation(Spec, Inputs, Outputs, Control\_Dependence, Data\_Dependence)$,其中:

1) $Spec$:$(ID, Name, Description)$为操作基本信息的定义(操作编号、操作名称及操作功能描述);

2) $Inputs:=\{data_i\}$为输入参数集合;

3) $Outputs:=\{data_i\}$为输出参数集合;

4) $Control_Dependence:=\{\langle ID, Dependence_i\rangle\}$为操作之间的控制依赖关系,包括顺序约束以及时间约束,定义执行顺序的各种约束条件;

5) $Data_Dependence:=\{\langle ID, Dependence_i\rangle\}$定义操作之间的数据依赖关系.

定义 4. 数据依赖. $\exists d_1, d_2, O_1, O_2$,且 $d_1\in O_1.Inputs\cup O_2.Outputs, d_2\in O_2.Inputs\cup O_2.Outputs$. 如果存在函数 F,使得 $d_2=F(d_1)$,则操作 O_1 和 O_2 存在数据依赖关系.

1) 如果 $d_1\in O_1.Inputs$ 并且 $d_2\in O_2.Inputs$,则操作 O_1 和 O_2 存在输入依赖关系,记为 $IND(O_1, O_2)$;

2) 如果 $d_1\in O_1.Outputs$ 并且 $d_2\in O_2.Outputs$,则操作 O_1 和 O_2 存在输出依赖关系,记为 $OUTD(O_1, O_2)$;

3) 如果 $d_1\in O_1.Outputs$ 并且 $d_2\in O_2.Outputs$,则操作 O_1 和 O_2 存在输入/输出依赖关系,记为 $INOUTD(O_1, O_2)$.

3　操作序列测试路径生成

3.1　操作序列测试流程

有状态 Web 服务含有多个操作,操作之间通过消息交互数据,因此仅仅对单个操作进行测试是不充分的,现有的针对操作序列测试的模型构建依赖于人员的经验,导致测试数据存在偏差,并且测试效率高. 为此,本文提出一种结合 EFSM 和接口契约模型的测试路径的生成方法,如图 2 所示.

测试路径的生成流程如下:

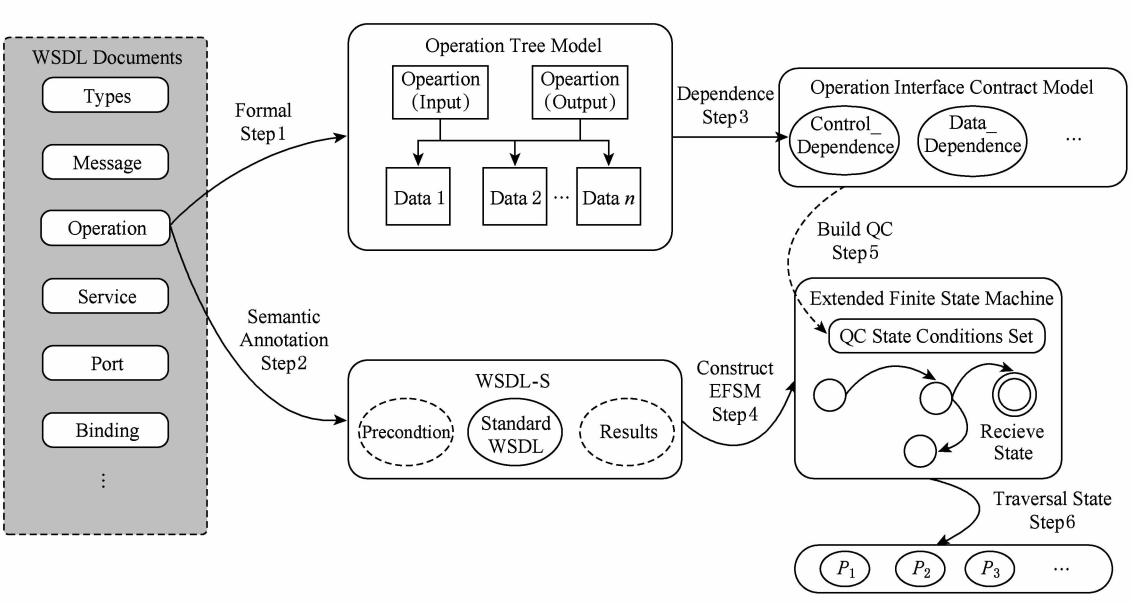

Fig. 2　Testing process of operation sequence

图 2　操作序列测试流程

Step1. 首先通过 Web 服务提供的标准 WSDL 获取操作输入输出参数等描述信息,建立操作树模型;

Step2. 根据 2.2 节的方法为 Web 服务中的操作添加语义标注(前置条件和结果);

Step3. 根据 Step1 建立的操作树模型,获取操作之间的依赖关系,构建操作接口契约模型,见第 3.2 节;

Step4. 基于扩展的语义 WSDL 文件为 Web 服务构建 EFSM 模型,见 3.3 节;

Step5. 采用接口契约模型中的控制依赖关系建立状态的条件集 QC;

Step6. 遍历 Step5 生成的 EFSM 模型,生成操作路径.

3.2 构建操作依赖关系

根据文献[17]定义的 3 个规则进行操作的依赖匹配,生成操作依赖关系的算法如下:

算法 2. 操作依赖生成算法.

输入:WSDL 文档;

输出:操作之间的控制、数据依赖关系集合 $Control\_Dependence$, $Data\_Dependence$;

初始化:$UncovOperation=\varnothing$; /* 未被遍历的操作集合 */

① 根据图 1,获取 WSDL 文档中操作的集合 $Operation$,取一个未标记的操作 $o_i\in Operation$ 作为当前操作,$UncovOperation=Opeartion-o_i$;

② 比较操作 o_1 的输出消息和未被标记的一个操作 $o_j\in Operation$ 的输入消息,判断是否存在输入/输出数据依赖关系 $INOUTD(o_i,o_j)$,若存在,加入集合 $Control\_Dependence$, $Data\_Dependence$;

③ While($UncovOperation=\varnothing$),重复执行步骤②,直到所有的未被标记的操作都被比较过;

④ 赋值 $UncovOperation=Opeartion-o_i$,比较操作 o_i 的输出消息和未被标记的一个操作 $o_j\in Operation$ 的输出消息,判断是否存在输入/输出数据依赖关系 $INOUTD(o_j,o_i)$,若存在,加入集合 $Control\_Dependence$, $Data\_Dependence$;

⑤ While($UncovOperation=\varnothing$),重复执行步骤④,直到所有的未被标记的操作都被比较过;

⑥ 标记当前操作 $mark(o_i)$;

⑦ 重复步骤①~⑥,直到所有的操作都被标记.

3.3 EFSM 的系统化构建算法

基于 2.3 节提出的有限状态机模型以及前序流程获取的语义 WSDL 和接口契约模型,本节根据文献[17,24]提出 EFSM 模型的系统化构建方法,具体如下:

1) 基于 WSDL 文档构建 Web 服务的行为信息表

定义 5. 行为信息表. 构建表 $T\subseteq O\times A\times\{`I',`S',``\}$, $O$ 和 $A$ 分别代表有状态 Web 服务中操作以及内部变量的集合, $\forall o\in O,\forall a\in A,(o,a,``)\in T$ 或 $(o,a,`I')\in T$ 或 $(o,a,`S')\in T$,对于 $\forall o\in O$,其中,`` 代表该操作无法获取或者无法更改的变量,$`S'$表示被该操作初始化的数据变量,$`I'$代表操作修改的数据变量, $I(o)=\{a\in A\mid(o,a,`I')\in T\}$表示被操作 $o$ 初始化的数据变量, $S(o)=\{a\in A\mid(o,a,`S')\in T\}$表示被操作 o 更改的数据变量.

2) 基于行为信息表构建 Web 服务的操作方式及其取值域

定义 6. 操作模式. 操作模式是一个由一系列数据变量集合$\{a_{k1},a_{k2},\cdots,a_{ks}\}$组成的抽象变量 e_k. 行为信息表含有 m 行和 n 列,操作分别为 $o_1,o_2,\cdots,o_m$,数据变量为 $a_1,a_2,\cdots,a_n$. 一个操作方式 $e_k=[a_{k1},a_{k2},\cdots,a_{ks}]$ 由一个变量 s 的数据集$\{a_{k1},a_{k2},\cdots,a_{ks}\}$组成,其中 $1\leqslant s\leqslant n$, $1\leqslant k1\leqslant k2\cdots\leqslant ks\leqslant n$,对于操作 $\exists o\in O$,它必须满足 2 个条件:

1) $\forall i(1\leqslant i\leqslant s),(o,a_{ki},``)\notin T$;

2) $\forall a_j\notin\{a_{k1},a_{k2},\cdots,a_{ks}\}$,当 $1\leqslant j\leqslant n$ 时,$(o,a_j,``)\in T$.

对于操作模式 e_k,满足条件 1)2)的操作集合行为 e_k 的操作等价类 $OE=\{oe_1,oe_2,\cdots,oe_s\}$

同时,操作行为的值域表示为一个操作行为的取值,它通过综合每一个操作行为在操作等价类中操作的变化的数据来获取,每一个行为 e_k 的值域为 V_k,通过语义信息 SWRL 来描述操作改变数据的结果.

3) 获取模式值的属性

当操作模式 $e_k=[a_{k1},a_{k2},\cdots,a_{ks}]$ 和 e_k 的取值为 V_k,模式值属性的定义如下.

定义 7. 模式值.

$v:O_u$ 是指当前模式值不能满足它相应操作的前置条件;

$v:O_n$ 是指能够满足前置条件的操作,$O=v:O_u\cup v:O_n$;

$v:O_c$ 表示能够对当前的值 v 进行改变为其他在 e_k 的值域的操作集合;

模式值转换函数 η:如果操作 o_i 改变模式 e_k 的值 v 到 w,若 $o_i\in v:O_c$,模式值转换函数为 $\eta(o_i,v,$

$precondition)=w$,其中 $w\in V_k$ 并且 precondition 是操作 o_i 的前置条件.

4) 获取 EFSM 的状态值

定义 8. 冲突模式值对.

取任意的操作模式 e_k 和 e_i,它们的值域分别为 V_k 和 V_i. 对于 $\forall v\in V_k$ 和 $\forall v'\in V_i$,如果满足 $v:O_n\cap v':O_u\neq\varnothing$ 或 $v:O_u\cap v':O_n\neq\varnothing$,$(v,v')$,则称为冲突模式对.

算法 3. 获取模型 EFSM 中的 Web 服务状态.

输入:Web 服务的操作模式 $E=\{e_1,e_2,\cdots,e_n\}$,相应的值域集 $V=\{V_1,V_2,\cdots,V_n\}$,操作集 $O=\{o_1,o_2,\cdots,o_m\}$;

输出:EFSM 的状态集 Q 和初始状态 q_0.

① 获取所有模式值可根据属性的条件构建冲突模式对的情况,得到所有冲突模式值对(v,v')的集合 C;

② 从可能的状态的集合 Q 中推算出所有模式值的组合,$Q=\prod_{i=1}^{n}V_i$;

③ 对于 $\forall q\in Q$,通过模式值形成的集合 q_v 构建 q,若 $v\in q_v$,$v'\in q_v$,并且$(v,v')\in C$,则 $Q=Q-\{q\}$;

④ 增加初始状态 q_0;

⑤ 返回状态集 Q 和初始状态 q_0.

5) 获取 EFSM 模型输入输出集合

设输入输出合集为 Σ,将所有操作的输入看作是 EFSM 模型的输入集合 I,所有操作的输出看作是 EFSM 模型的输出集合 O,每一个输入输出表示为一类数据类型.

6) 获取 EFSM 模型的转移函数 $\Phi:Q\times\Sigma\times QC\longrightarrow Q$

算法 4. 状态转移函数获取算法.

输入:操作集合 O,模式集合 $E=\{e_1,e_2,\cdots,e_n\}$,模式值 $e_1,e_2,\cdots,e_n$,状态集合 Q;

输出:状态转移函数 φ.

① $F=\varnothing$. /* 初始化函数集 */

② 对于初始状态 q_0,若状态 $q=(v_1,v_2,\cdots,v_{|E|})\in Q$,对于每一个模式值 v_i,满足条件 $v_i=\{I,I,\cdots,I\}$ 或 $v_i=\{X,X,\cdots,X\}$,当 I 或 X 在 v_i 的取值之间,那么 $\varphi(q_0,o)=q$ 存在,o 是起始执行操作,它包含了数据变量的初始化.

③ 对于每一个状态 $q=(v_1,v_2,\cdots,v_{|E|})\in Q-\{q_0\}$,执行下面步骤获取它的初始转换:

Ⅰ 无转换

若 $q:O_u=\bigcup_{v\in V}v:O_u$,$\forall o\in q:O_u$,则 $\varphi(q_0,o)=q$ 没有被定义.

Ⅱ 转换到其他状态

$$q:O_c=\bigcup_{v\in V}v:O_c-q:O_u.$$

$\forall o\in q:O_c$,则 $\varphi(q_0,o)=w$,设 $w=(w_1,w_2,\cdots,w_{|E|})$:

如果 $o\in v_i:O_c$ $(1\leqslant i\leqslant n)$,则 $w_i=\eta(o,v_i,precondition)$ 或 $w_i=\eta(o,v_i,\varepsilon)$;

如果 $o\notin v_i:O_c(1\leqslant i\leqslant n)$,则 $w_i=v_i$.

Ⅲ 环状态转换

令 $q:O_l=O-q:O_c-q:O_u$,$\forall o\in q:O_l$,$\varphi(q_0,o)=q$.

④ 返回状态转换函数 φ.

7) 遍历构建模型生成操作序列的测试路径

生成的 EFSM 模型如图 3 所示. 以上流程除了步骤 1)2),其余步骤均可自动化执行,相比于手工构建 Web 服务测试模型,该方法节省了测试成本,并且避免了人为操作带来的错误.

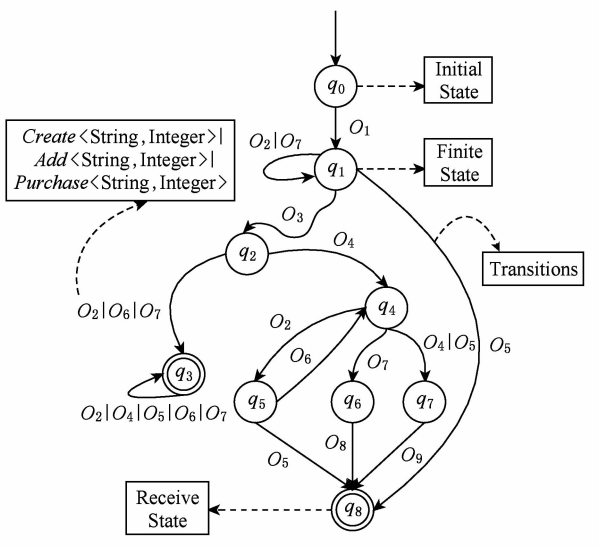

Fig. 3　Diagram of state transition for EFSM model

图 3　EFSM 状态转移图

4　基于操作接口契约模型的测试数据生成

4.1　基于接口契约及数据分区的测试数据生成

根据 3.3 节的算法能够自动化获取有状态 Web 服务的操作序列测试路径,可以根据常规的方法逐个为路径中的操作生成测试数据验证测试序列的正确性. 可是这样的生成方法含有 2 点缺陷:

1) 测试路径中测试数据生成方法没有考虑各操作之间的输入/输出依赖关系,例如在某个测试路径中,上一个操作 o_i 的输出可能是下一个操作 o_j 的输入(其中 i 和 j 表示操作在测试路径中出现的索引,$j>i$),那么操作 o_j 的测试数据就可以从 o_i 的输出获取,若采用常规方法提取,则会增加了生成数据的工作量.

2) 测试路径中某操作获取测试数据时,要考虑到保证该测试路径中后续操作都存在合适的输入数据,以保证该测试路径的执行.

针对上述缺陷,本文首先采用数据分区的方法,对测试数据进行设计,根据输入数据和全局变量等参数将数据划分为多个子区域[35-36]. 之后对操作的输入输出数据进行数据分区合法性校验,以保证测试路径的正常执行,提高测试数据设计的合理性. 其次,我们根据操作接口契约模型中的数据依赖关系,对测试序列输入输出数据进行匹配,减少不必要的测试数据生成.

4.2 用例集生成及优化算法

算法 5. 用例集生成算法.

输入:EFSM 生成的测试路径集合 $O=\{P_1(o_1,o_2,o_5),P_2(\cdots),\cdots,P_n(\cdots)\}$ 及接口契约模型 $Operation\langle\rangle$;

输出:初始测试数据集 $Testdata_0$.

```
for 集合 O 中的每一个集合 P_i
  O={P_1(o_1,o_2,o_5),P_2(…),…,P_n(…)}
/*获取路径组合中每一条测试路径*/
  for 集合 P_n 中的每一个参数 o_i/*遍历测试路径中每一个操作*/
    I=∅; /*置空输入参数集合 I*/
    if(o_i.pre=Null){
      Generate(o_i.input); /*采用等价类、边界值划分方法为路径中初始操作 o_i 生成数据*/
      Execute(o_i.input,o_i.output);} /*执行操作,并存储操作 o_i 输出数据*/
    else if(o_i.input≠Null) /*若 O_i 的输入不为空*/{
      Dependency(o_i.input,o_{i-1}.output);
        /*将操作 o_i 的全部输入参数数据放入集合 I,并在操作 o_i 中标记,根据操作的接口依赖关系将匹配上的前序操作 o_{i-1} 的输出数据匹配为 o_i 的输入数据*/
      Generate(I-o_{j-1}.output);} /*采用等价类、边界值方法为路径中操作 o_i 生成未匹配的输入数据数据*/
      Execute(o_i.input,o_i.output);} /*执行存储操作 o_i 输出数据*/
    else{Execute(o_i.input,o_i.output);}
      /*执行存储操作 o_i 输出数据*/
    endif
  end for
end for
```

算法 6. 用例集优化算法.

输入:测试数据集 $Testdata_0$;

输出:优化用例集 $Testdata$.

```
While(o_i.next∈P_n){
  /*判断是否遍历到结尾*/
  GenerateDataFor(P_n); /*调用算法 5 为测试路径 P_n 生成初始测试数据*/
  for 集合 P_n 中的每一个参数 o_i:
    if (o_i.input∉Par^{o_i}_{valid}){/*判断当前操作 o_i 的输入数据是否在 o_i 的合法分区内,这里的分区包含所有参数的分区比较*/
      Break;} /* 若不属于,重新根据路径指向初始数据算法*/
    end if
  end for}
```

输出优化后的能够执行路径的测试数据,能够保证测试路径的执行.

5 实例分析

本文选择某型号面向服务的舰船指控系统软件作为待测对象,选取任务执行模块的软件子功能作为测试对象,服务包含操作(StartRecordTask,QuitRecordTask,SuspendReplayTask 等). 这类操作对于同样的输入参数得到不同的执行结果,服务是有状态的,因而需要进行操作序列的测试. 在实验过程中,我们依据 3.1 节描述的流程,首先,基于服务描述文件 WSDL 构建操作树模型并进行语义标注,之后通过行为信息构建 EFSM 模型,进而遍历得到操作序列测试路径,最终根据接口契约模型中

的依赖关系生成初始测试数据,并采用数据分区的方法优化测试用例数据.

实验中使用 SOAP 消息对测试用例进行封装,并通过 HTTP 进行数据传输,将 SOAP 请求发送到该服务所在的服务器,处理流程包括测试用例的生成选择、测试执行和测试结果分析.

5.1　覆盖率分析

本节进行了对比实验,实验中随机生成 600 个测试用例,用工具 SoapUI 运行后发现,部分用例由于随机生成的数据元素存在盲目性,导致实验中生成的操作序列路径无效,无法返回结果.之后,在生成的测试数据中分别选取 50,100,150,200,250,300,350,400,450,500 个数据组成 10 个不同大小的测试集合,分别使用软件测试工具 McCabe IQ (V8.0)对测试模块进行插桩,将插桩生成的源程序一条编译连接,执行功能用例执行,监视和获取服务的代码覆盖情况,通过与随机测试用例的覆盖率情况相比较,本文提出的方法均获得了更好的代码和语句覆盖率,具体如图 4、图 5 所示:

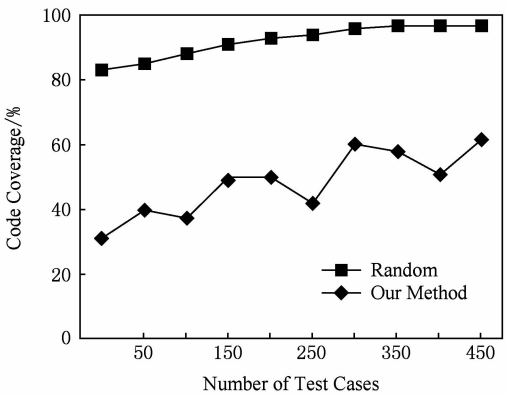

Fig. 4　Code coverage of each test suite

图 4　代码覆盖率对比情况

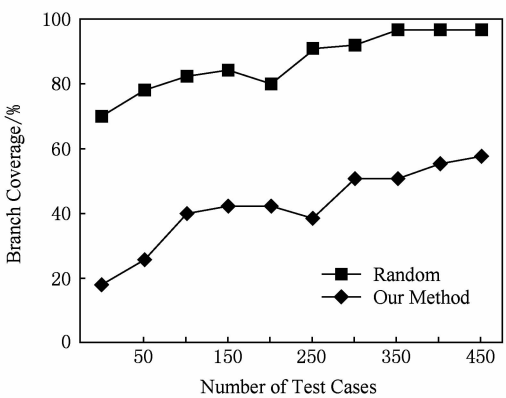

Fig. 5　Branch coverage of each test suite

图 5　分支覆盖率对比情况

根据服务需求设计文档构建业务流程剖面,获取指控执行模块服务中存在的可执行操作序列的路径为 18 条,而每条操作序列则需要根据 EFSM 的状态迁移建立多个用例测试路径.本文对不同数量的测试用例集进行服务操作序列的覆盖充分性分析和计算,并与随机生成的测试用例进行比较,覆盖情况如表 1 所示:

Table 1　Operation Sequence Coverage Adequacy

表 1　操作序列覆盖充分性

Number of Test Cases	Random		Our Method	
	Coverage Path	Path Coverage/%	Coverage Path	Path Coverage/%
100	3	16.6	6	33.3
200	5	22.2	10	55.5
300	8	44.4	13	72.2
400	8	44.4	18	100
500	10	55.5	18	100

5.2　用例错误检测能力分析

为了证明本文方法在错误检测能力以及用例规模上的优势,通过现有的方法对实验中的待测服务中存在的操作序列进行测试数据的生成,并根据执行结果进行比较,结果如图 6 所示.此外,表 2 给出了各方法执行过程中多个操作的相关结果.

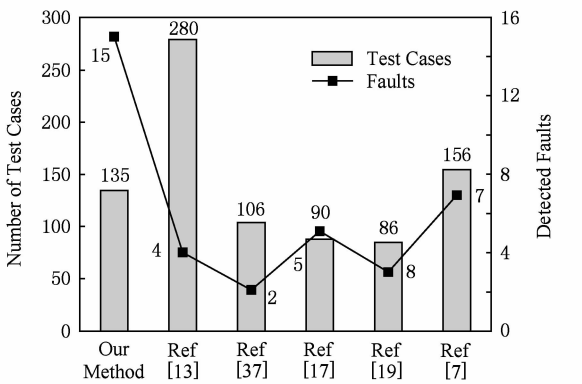

Fig. 6　Number of test cases and number of faults

图 6　测试用例数量及发现错误数

如我们预期,本文的方法用较少的用例发现了更多的错误,表明了生成的测试用例拥有较强的针对性.文献[13,36]的方法致力于生成单个操作的测试用例,对于有状态的 Web 服务很少有错误是能够通过单个操作的边界值测试方法检测出的,因而仅仅依赖于单个操作生成测试数据的方法是不充分的.虽然文献[17,19]也提出了运用数据流的方法加

强测试序列的测试,但由于该方法只考虑了少量偶然的操作之间的关系,导致生成用例的数量不够,降低了用例的错误检测能力. 文献[7]提出使用模型的方法,可是由于模型的生成依赖于测试人员的经验,生成的数据具有严重的不平衡性,降低了测试效率,且没有考虑到数据之间的依赖关系,简单地的为每一个操作生成测试数据,会导致用例规模过于庞大,无法满足测试需求. 因此,本文提出的方法由于考虑了复杂操作序列的交互过程,在测试用例集规模和错误检测能力上均占有一定的优势.

Table 2 Test Cases and Results

表 2 测试用例及结果

Data Generation Method	Test Cases	Detected Faults	Fault Description	No Results
Our Method	135	15	Boundary exceptions, Null value and Operation interact exceptions faults etc	2
Ref[13]	280	4	Function faults for single operation	65
Ref[37]	106	2	Partly Boundary exceptions,	26
Ref[17]	90	5	Partly Operation interact exceptions	10
Ref[19]	86	3	Partly Function faults	15
Ref[7]	156	7	Partly Function faults and Operation interact exceptions faults etc	8

此外,据表 2 所述,实验中有些用例无法得到执行结果,这反映了在生成测试路径测试数据时,没有考虑到测试路径的可行性,有些数据可能在路径之中就由于不合法,导致路径无法执行,无法返回预期结果. 而这些没有返回正确结果的用例,输入参数往往符合数据类型的需求,但提交给服务时,输出却是无法返回结果,这将难以判断是由于用例设计有误所导致的后果还是 Web 服务本身就存在问题. 本文提出的方法根据数据分区的方法增强了测试数据的有效性,以保证测试路径的执行,达到设计的用例可行有效,较少无返回结果.

6 测试用例自动生成原型系统

结合研究团队之前的 Web 服务测试用例集生成的相关工作,本文研制了 Web 服务测试用例自动生成原型系统 SOATest. 本工具支持 Web 服务的功能测试和性能测试. 具体功能包括测试用例的生成和管理、测试的执行,以及测试结果的收集和报表生成. 工具支持以 WSDL 为接口描述文件的标准服务实体的单个操作以及操作序列的测试用例生成,测试系统的体系结构如图 7 所示:

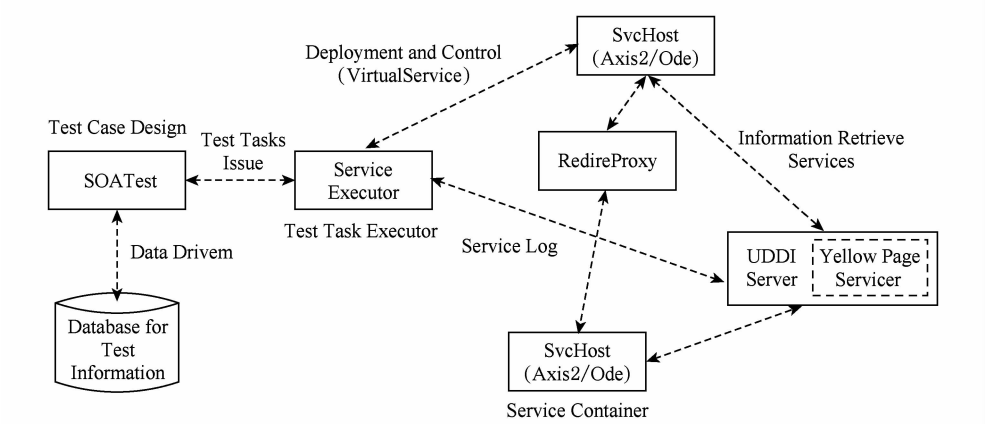

Fig. 7 Test generation prototype system architecture

图 7 测试用例自动生成原型系统体系结构

运用工具 SOATest 的服务单体测试模块可以为 Web 服务单个操作生成用例,运用工具服务序列测试模块可以为有状态 Web 服务的操作序列自动生成测试用例. 本系统首先识别服务中的所有操作,然后通过路径算法生成服务操作的全排列并可对生成的测试用例进行管理. 基于接口契约的方式根据状态迁移过滤无效序列,构建操作序列流程图,并依此生成有效测试数据,执行、获取测试结果,系统执

行界面如图 8～11 所示,分别从 Web 服务的操作序列的生成、测试用例的生成、测试用例的管理及执行及用例执行结果图等多个方面对原型系统进行展示.

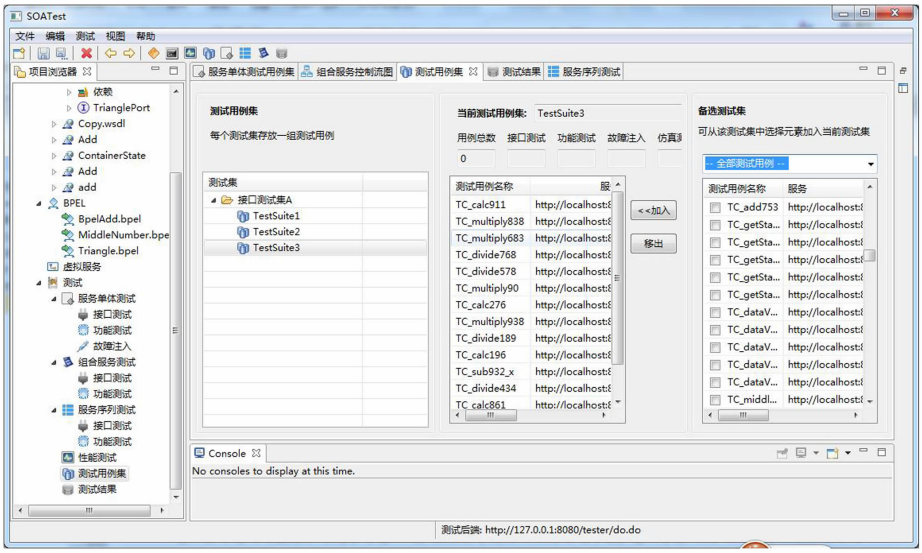

Fig. 8　Interface of test cases management

图 8　测试用例管理界面

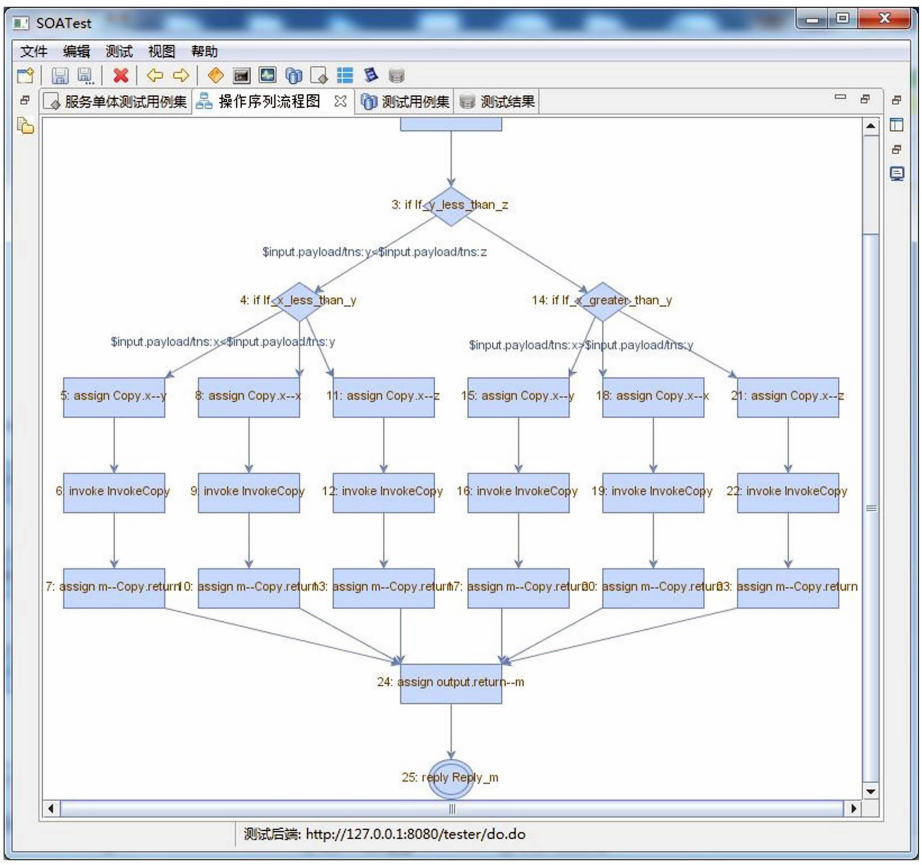

Fig. 9　Interface of operation process

图 9　操作序列流程界面

表 3 给出了运用工具为某型号面向服务的舰船指控系统软件中的 5 个案例有状态服务生成测试路径的相关信息,包含了案例对应的状态数,生成的路径数、生成的路径的长度及路径计算耗时. 实验表

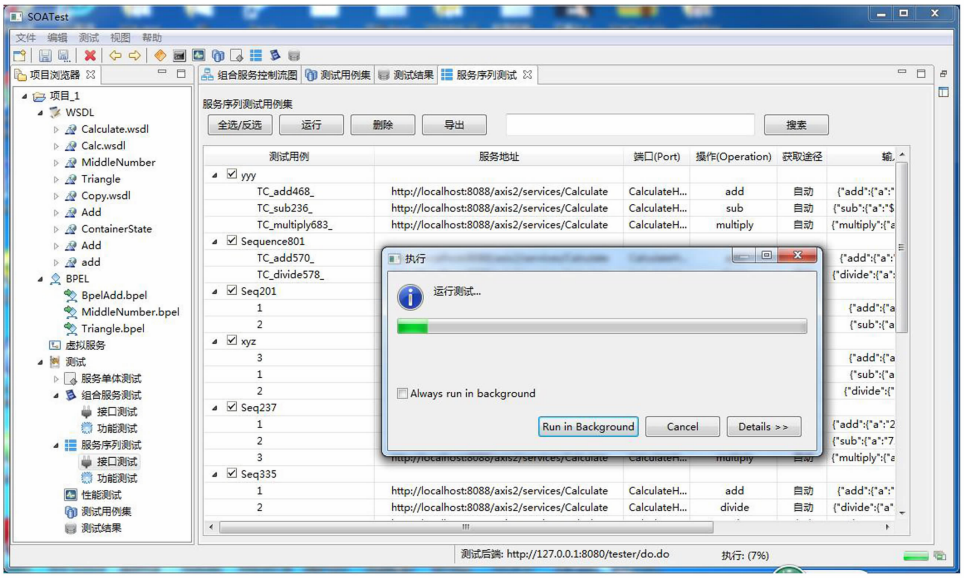

Fig. 10 Interface of test execution

图 10 测试用例执行界面

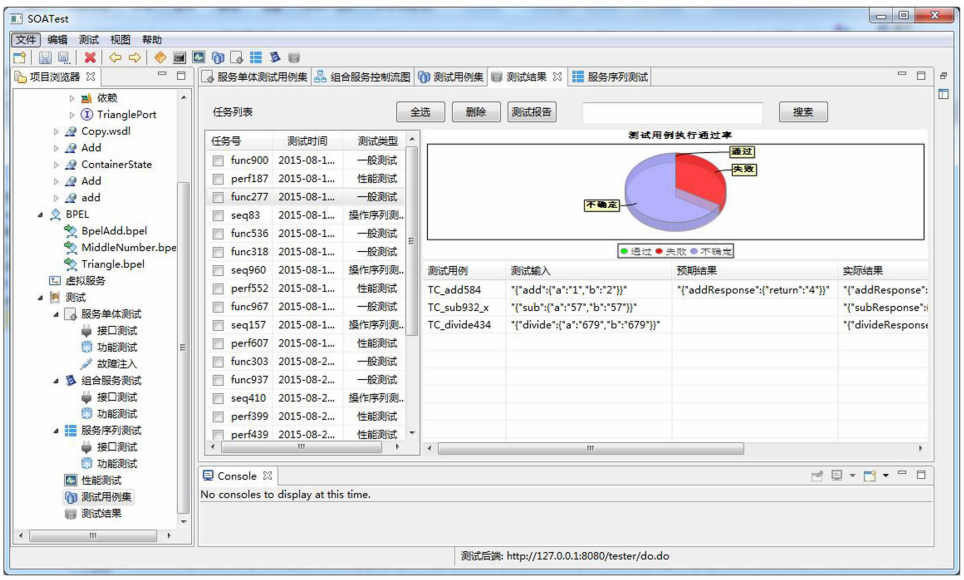

Fig. 11 Interface of results of test cases

图 11 测试用例执行结果界面

Table 3 Test Cases and Results

表 3 案例程序的测试路径集的信息统计

Service Case	States	Total Paths	The Longest Length of Path	Execute Time/ms
Show Information Service	5	22	4	420
Business Process Service	7	42	5	90
Task Manage Service	16	116	9	1 485
User Manage Service	8	60	6	210
Information Processing Service	6	30	11	135

明:本原型系统能够正确有效地生成有状态服务的操作序列测试用例集,且计算耗时与各 EFSM 的状态数相关.

7　结束语

Web 服务的动态查找、调用等机制也给测试带来了很多难题,如何确保 Web 服务软件的质量和可靠性成为了当前软件测试领域的研究热点之一.目前虽然存在一些 Web 服务测试技术方面的研究,但是由于标准的 WSDL 文件缺少一些语义、动态以及行为信息的描述,使得测试通常仅仅采用 Web 服务的句法信息,这也导致现有的研究更多地关注于测试单个操作而非操作序列的测试.此外,现有的 Web 服务测试方法更依赖于工程师的经验,降低了测试的均衡性以及自动化程度.

在现有工作基础上,本文提出了一种有状态 Web 服务的测试数据生成方法:1)基于标准的 WSDL 进行操作树模型构建以及语义标注;2)根据增强的行为信息,通过有限状态机模型(EFSM)描述服务的动态行为;3)根据接口契约模型生成单个 Web 服务操作序列的测试数据,并通过实例验证了该方法在测试用例集和错误检测能力方面的优越性,解决了之前研究无法有效地自动生成操作序列数据等问题,增强了测试路径生成的自动化程度和测试数据的有效性,从而提高了 Web 服务测试数据自动生成的质量和效率.

在后期的工作中,我们将针对模型通用性以及算法效率进行详尽的研究,并完善相应的原型工具更好地推广提出的技术框架,同时计划将该方法扩展应用到服务组合测试中,进一步推进 Web 服务的自动化测试.

参　考　文　献

[1] Mei Hong, Zhang Lu. A framework for testing Web services and its supporting tool [C] //Proc of IEEE SOSE'05. Los Alamitos, CA: IEEE Computer Society, 2005: 199-206

[2] Bartolini C, Bertolino A, Marchetti E, et al. WS-TAXI: A WSDL-based testing tool for Web services [C] //Proc of IEEE ICST'09. Los Alamitos, CA: IEEE Computer Society, 2009: 326-335

[3] Xu Lei, Li Yanhui, Chen Lin, et al. A testing method for Web services focusing on user requirements [J]. Chinese Journal of Computers, 2014, 37(3): 512-521 (in Chinese)
(许蕾,李言辉,陈林,等. 一种面向用户需求的 Web 服务测试方法[J]. 计算机学报, 2014, 37(3): 512-521)

[4] Elia I A, Laranjeiro N, Vieira M. A practical approach towards automatic testing of Web services interoperability [J]. International Journal of Web Services Research, 2015, 12(3): 103-129

[5] Brenner D, Atkinson C, Hummel O, et al. Strategies for the run-time testing of third party Web services [C] //Proc of IEEE SOCA'07. Piscataway, NJ: IEEE, 2007: 114-121

[6] Sinha A, Paradkar A. Model based functional conformance testing of Web services operating on persistent data [C] //Proc of 2006 Workshop on Testing, Analysis, and Verification of Web Services and Application. New York: ACM, 2006: 17-22

[7] Keum C S, Kang S, Ko I Y, et al. Generating test cases for Web services using extended finite state machine [G] //LNCS 3964: Proc of the 18th IFIP TC 6/WG 6.1 Int Conf (TestCom 2006). Berlin: Springer, 2006: 103-117

[8] Ma Chunyan, Zhou Yian, Lu Wei. Automatic test for Web services [J]. Computer Science, 2012, 39(2): 162-169 (in Chinese)
(马春燕,朱怡安,陆伟. Web 服务自动化测试技术[J]. 计算机科学, 2012, 39(2): 162-169)

[9] Zhong Li, Sun Jie, Jiang Wei, et al. BPEL4WS unit testing: Framework and implementation [C] //Proc of the 2005 Int Conf on Web Services(ICWS'05). Los Alamitos, CA: IEEE Computer Society, 2005: 103-110

[10] Tsai W T, Paul R, Wang Y, et al. Extending WSDL to facilitate Web services testing [C] //Proc of the 7th IEEE Int Symp on High Assurance Systems Engineering. Los Alamitos, CA: IEEE Computer Society, 2002: 171-172

[11] Xu Wuzhi, Offutt J, Luo Juan. Testing Web services by XML perturbation [C] //Proc of the 16th IEEE Int Symp on Software Reliability Engineering. Los Alamitos, CA: IEEE Computer Society, 2005: 256-266

[12] Hanna S, Munro M. An approach for specification-based test case generation for Web services [C] //Proc of 2007 IEEE ACS Int Conf on Computer Systems and Application. Los Alamitos, CA: IEEE Computer Society, 2007: 16-23

[13] Ma Chunyan, Du Chenglie, Zhang Tao, et al. WSDL-Based automated test case generation for Web service [C] //Proc of the Computer Science and Software Engineering. Los Alamitos, CA: IEEE Computer Society, 2008: 731-737

[14] Siblini R, Mansour N. Testing Web services [C] //Proc of IEEE ISSRE'05. Los Alamitos, CA: IEEE Computer Society, 2005: 763-770

[15] Jiang Ying, Xin Guomao, Shan Jinhui, et al. A method of automated test data generation for Web service [J]. Chinese Journal of Computers, 2005, 28(4): 568-577 (in Chinese)
(姜瑛,辛国茂,单锦辉,等. 一种 Web 服务的测试数据自动生成方法[J]. 计算机学报, 2005, 28(4): 568-577)

[16] Bai Xiaoying, Dong Wenli, Tsai W-T, et al. WSDL-Based automatic test case generation for Web services testing [C] // Proc of the 2005 IEEE Int Workshop on Service-Oriented System Engineering. Los Alamitos, CA: IEEE Computer Society, 2005: 207-212

[17] Li Li, Wu Chou. Automatic message flow analyses for Web services based on WSDL [C] //Proc of 2007 IEEE Int Conf on Web Services. Los Alamitos, CA: IEEE Computer Society, 2007: 25-28

[18] Bertolino A, Polini A. The audition framework for testing Web services interoperability [C] //Proc of the 31st EUROMICRO Conf on Software Engineering and Advanced Application. Los Alamitos, CA: IEEE Computer Society, 2005: 134-142

[19] Heckel R, Mariani L. Automatic conformance testing of Web services [G] //LNCS 3442: Proc of the 8th Int Conf FASE 2005. Berlin: Springer, 2005: 34-48

[20] Belli F, Linschulte M. Event-driven modeling and testing of Web services [C] //Proc of IEEE COMPSAC'08. Los Alamitos, CA: IEEE Computer Society, 2008: 1163-1173

[21] Paradkar A, Sinha A, Williams C, et al. Automated functional conformance test generation for semantic Web services [C] //Proc of IEEE ICWS'07. Los Alamitos, CA: IEEE Computer Society, 2007: 110-117

[22] Sinha A, Paradkar A. model-based functional conformance testing of Web services operating on persistent data [C] // Proc of the 2006 Workshop on Testing Analysis and Verification of Web Services and Applications. Los Alamitos, CA: IEEE Computer Society, 2006: 17-22

[23] Paradkar A M. Automated functional conformance test generation for semantic Web services [C] //Proc of 2007 IEEE Int Conf on Web Services. Los Alamitos, CA: IEEE Computer Society, 2007: 110-117

[24] Ma Chunyan, Wu Junsheng, Zhang Tao. Web services sequence testing based on stream X-machine [C] //Proc of the 10th Int Conf on Quality Software. Los Alamitos, CA: IEEE Computer Society, 2010: 232-239

[25] Bai Xiaoying, Lu Hao, Zhang Yao, et al. Interface-based automated testing for open software architecture [C] //Proc of IEEE COMPSACW'11. Los Alamitos, CA: IEEE Computer Society, 2011: 149-154

[26] Petrova A, Dessislava I, Denitsa M, et al. TASSA: Testing framework for Web service orchestrations [C] //Proc of the 10th Int Workshop on Automation of Software Test in Conjunction with the 37th Int Conf on Software Engineering. Los Alamitos, CA: IEEE Computer Society, 2015: 8-12

[27] Vanderveen P, Janzen M, Tappenden A F. A Web service test generator [C] //Proc of 2014 IEEE Int Conf on Software Maintenance and Evolution. Los Alamitos, CA: IEEE Computer Society, 2014: 516-520

[28] W3C. Web service description language (WSDL) version 2.0 part1: Core language [EB/OL]. (2007-06-26) [2016-05-23]. http://www.w3.org/TR/wsdl20

[29] Belhajjame K, Embury S M, Paton N W. Verification of semantic Web service annotations using ontology-based partitioning [J]. IEEE Trans on Services Computing, 2014, 7(3): 515-528

[30] He Lingjuan, Liu Lianchen, Wu Cheng. A modified operation similarity measure method based on WSDL description [J]. Chinese Journal of Computers, 2008, 31(8): 1331-1339 (in Chinese)
(何玲娟, 刘连臣, 吴澄. 一种改进的基于 WSDL 描述的操作相似性度量方法[J]. 计算机学报, 2008, 31(8): 1331-1339)

[31] Kunal M. WSDL-S: Adding Semantics to WSDL-WhitePaper [EB/OL]. (2007-06-26)[2016-05-23]. http://www.w3.org/TR/2003/, 2003-04-01John

[32] Miller J, Verma K, Rajasekaran P, et al. WSDL-S: Adding semantics to WSDL-White paper [EB/OL]. (2004-01) [2016-05-23]. http://lsdis.cs.uga.edu/library/download/wsdl-s.pdf

[33] Horrocks I. SWRL: A semantic Web rule language combining owland ruleML [EB/OL]. (2004-05) [2016-05-23]. http://www.w3.org/submission/2004/SUBM-SWRL-20040521

[34] Hou Kejia, Bai Xiaoying, Lu Hao, et al. Web service test data generation using interface semantic contract [J]. Journal of Software, 2013, 24(9): 2020-2041 (in Chinese)
(候可佳, 白晓颖, 陆皓, 等. 基于接口语义契约的 Web 服务测试数据生成[J]. 软件学报, 2013, 24(9): 2020-2041)

[35] Ostrand T J, Baiter M J. The category-partition method for specifying and generating functional tests [J]. Communications of the ACM, 1988, 31(6): 676-686

[36] Chen T Y, Poon P, Tse T H. A choice relation framework for supporting category-partition test ease generation [J]. IEEE Trans on Software Engineering, 2003, 29(7): 577-593

[37] Xu Wuzhi, Offutt J, Luo Juan. Testing Web services by XML perturbation [C] //Proc of the 16th IEEE Int Symp on Software Reliability Engineering. Los Alamitos, CA: IEEE Computer Society, 2005: 257-266

Li Yin, born in 1988. Master. His main research interests include data mining and software testing.