一种基于合约式设计的测试技术研究*

姜 瑛[1,2+], 辛国茂[1], 单锦辉[1], 谢 冰[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,云南 昆明 650093)

Research on a Testing Technology Based on Design-by-Contract

JIANG Ying[1,2+], XIN Guo-Mao[1], SHAN Jin-Hui[1], XIE Bing[1]

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

[2](Faculty of Information Engineering and Automation, Kunming University of Science and Technology, Kunming650093, China)

+ Corresponding author: Phn: +86-10-62751794, E-mail: jiangy@sei.pku.edu.cn, http://www.pku.edu.cn

Received 2004-01-16; Accepted 2004-03-29

Jiang Y, Xin GM, Shan JH, Xie B. Research on a testing technology based on design-by-contract. *Journal of Software*, 2004,15(Suppl.):130~137.

Abstract: In order to test Web Services, a testing technology based on Design-by-Contract is proposed in this paper. Usually, there are two problems when the users are using Web Services. Firstly, they cannot locate the occurring errors during testing precisely. Secondly, it is difficult to test whether Web Services meet their requirements. Design-by-Contract is an effective method to improve software reliability. This paper proposes a testing technology for Web Services based on preconditions and postconditions, then extends the syntax of WSDL and implements a prototype on the Microsoft .NET platform, which can solve the above problems better.

Key words: Web services; software testing; design-by-contract; precondition; postcondition

摘 要: 所研究的基于合约式设计的测试技术是针对 Web Services 的.在使用 Web Services 时通常会遇到两个问题:① 无法准确定位测试时出现的错误;② 很难测试 Web Services 是否符合使用者的需求.合约式设计是一种提高软件可靠性的有效方法.结合合约式设计,提出了一种基于前置条件和后置条件的 Web Services 测试技术,对 WSDL 语法进行了扩展,并在 Microsoft .NET 平台上实现了原型,较好地解决了以上问题.

关键词: Web Services;软件测试;合约式设计;前置条件;后置条件

Web Services 是一种新型的中间件技术,根据 W3C 对 Web Services 的定义[1],Web Services 以 SOAP(Simple

*. Supported by the National Natural Science Foundation of China underGrant No.60373003 (国家自然科学基金); the National High-Tech Development 863 Program of China under Grant No.2001AA113070 (国家高技术研究发展计划(863)); the National Grand Fundamental Research 973 Program of Chinaunder Grant No.2002CB31200003 (国家重点基础研究发展规划(973)); China Postdoctoral Science Foundation under Grant No.2003034077(中国博士后科学基金资助项目)

作者简介: 姜瑛(1974－),女,浙江余姚人,博士生,主要研究领域为软件工程,软件测试,软件构件技术;辛国茂(1981－),男,硕士生,主要研究领域为软件测试,软件构件技术;单锦辉(1970－),男,博士,主要研究领域为软件测试,构件技术,形式化方法;谢冰(1970－),男,博士,副教授,主要研究领域为软件工程,形式化方法,分布式系统.

Object Access Protocol)、WSDL(Web Services Description Language)和 UDDI(Universal Description, Discovery and Integration)为核心,致力于解决软件在应用层的互操作问题[2].Web Services 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,适合构造松耦合的软件体系结构.因此,Web Services 获得了产业界广泛的支持和学术界的重视[3].

需要说明的是,Web Services 通常指代原理和技术,Web Service 则指代具体的应用实例,而多个应用实例也称为 Web Services,因此需要根据不同的上下文来判断 Web Services 的具体含义.在本文中,凡是由服务提供者提供、服务使用者使用和测试的 Web Service 和 Web Services 均指应用实例.

Web Services 体系结构模型基于服务提供者、服务注册中心和服务使用者三种角色之间的交互[4].服务提供者在开发和验证自己的 Web Services 时,不可能预先设想这些 Web Services 所有可能的使用环境.因此,Web Services 的测试和评估对服务使用者非常重要.

然而,测试 Web Services 很困难,其原因在于:服务使用者需要选择和调用 Web Services,在使用中还涉及到 UDDI、SOAP、WSDL 等标准协议;Web Services 是包含大量运行时行为的分布式应用,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 Services 的设计或实现细节,那么服务使用者能够获得的通常只有 Web Services 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.

WSDL 是 IBM 的 NASSL(network accessible services specification language)和 Microsoft 的 SDL(service description language)等基于 XML 的接口描述语言的发展,它描述了 Web Services 的功能规约,包括数据类型、操作的抽象描述、绑定信息、网络位置等,也可提供分类等元数据以便于服务使用者发现和使用 Web Services.当前的 WSDL 文档主要包括以下信息:① 输入/输出参数的数量;② 输入/输出参数的变量类型;③ 输入/输出参数的顺序;④ Web Services 的访问地址.由于 WSDL 文档只包含 Web Services 接口的抽象描述而没有接口的语义信息,因此给服务使用者使用和测试 Web Services 带来了不便.

通常,服务使用者在使用 Web Services 的过程中会遇到如下的问题:一是无法完全了解 Web Services 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地定位,即服务使用者无法分辨究竟是 Web Services 本身的错误,还是使用 Web Services 的方法有误;二是很难测试 Web Services 是否符合使用需求.文献[5,6]提出在原有的 WSDL 文档中加入 Web Services 输入/输出参数的依赖关系、调用顺序、层次功能描述和并发顺序规范等信息,主要关注对 Web Services 自身功能的测试,并不能完全解决以上问题,而且要求服务提供者提供大量的信息,这可能会给不愿意暴露过多 Web Services 细节的服务提供者带来困扰.本文借鉴合约式设计的思想,提出一种基于前置条件和后置条件的 Web Services 测试技术,在 WSDL 文档中增加服务提供者和服务使用者的合约,结合服务提供者和服务使用者的力量,以较低的开销解决上述两个问题.

1 合约式设计

一般来说,基于 Web Services 的软件开发中至少包含两个群体,一个群体是服务的提供者,另一个群体是服务的使用者.这两个群体有以下主要特点:① 服务提供者书写 Web Services 的源代码、提供代码的文档并维护代码和文档,他了解 Web Services 的实现并对外发布其方法的接口;② 服务使用者在自己的代码中使用 Web Services 的方法,他可以获得接口,但不了解其具体实现.合约式设计[7](Design-by-Contract,简称 DbC)就是要在上述两个群体之间建立合约.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 Services 所需的条件和 Web Services 能够完成的功能.常见的表达合约的种类有:前置条件、后置条件、不变式、循环变式和循环不变式等.

例如,假设有一个求平方根的方法 Square(num),该方法中定义的合约如下所示:

```
precondition:    num >= 0;
postcondition:   num - result * result < 0.0001;
```

前置条件表示该方法正确运行所需的条件或限制,如方法 Square 只能对非负数求平方根.后置条件表示该方法正确运行后的状态.

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;如果后置条件被违反,责任在提供者.文献[8]认为,合约使得可复用的构造块(building blocks)更易于被实现、测试和组装.因此,DbC 被视为提高软件可靠性的一种有效方法[7].

DbC要求服务提供者所开发的Web Services要达到合约中声明的功能,而服务使用者要满足合约中规定的使用该 Web Services 所需的条件.采用 DbC 方法,利用自动化的合约检查工具,服务提供者可以不必在 Web Services 的源代码中对其使用条件显式地进行检查,服务使用者不必对 Web Services 的运行结果进行检查,从而有助于减少冗余的检查代码,提高软件设计的效率和运行的性能.

2 基于 DbC 的 Web Services 测试

我们结合合约式设计,扩展了 WSDL 语法,提出了一种基于前置条件和后置条件的 Web Services 测试技术.

2.1 带有合约的WSDL

服务使用者可以从服务注册中心获得 Web Services 的接口信息——WSDL 文档.当前的 WSDL 文档中不包含合约,我们试图在 WSDL 中增加对 Web Services 中方法前置条件和后置条件的描述,对 Web Services 的功能进行更进一步的说明,让服务使用者获得关于该 Web Services 的更多的信息.由于 WSDL 文档中的 operation 部分是对 Web Services 对外提供的方法的抽象描述,因此我们把描述 Web Services 功能的合约放在 WSDL 文档中的 Operation 定义部分.下面是我们用 BNF 范式定义的合约的语法规则:

```
PortTypeSection ::= "<portType>" {OperationSection} "</portType>"
OperationSection ::= "<operation>" [InputSection] [OutputSection]{ContractSection}
                     "</operation>"
ContractSection::="<contract>" {PreconditionSection} {PostconditionSection} "</contract>"
PreconditionSection ::= "<precondition>" PreconditionExpression "</precondition>"
PreconditionExpression ::="(" PreconditionExpression ")" | PreExpression Comparator PreExpression |
                           UnaryConnector PreconditionExpression |
                           PreconditionExpression Connector PreconditionExpression
PreExpression ::= PreArithmaticExpression | PreLogicalExpression
PreArithmaticExpression ::= "(" PreArithmaticExpression ")" |  PreconditionOperator |
                            UnaryArithmaticOperator PreExpression |
                            PreExpression DualArithmaticOperator PreExpression
PreconditionOperator ::= ParameterName | PreconditionOperator"[" PreArithmaticExpression "]" |
                         PreconditionOperator "." FieldName | LITERAL
PreLogicalExpression ::= "(" PreLogicalExpression ")" | PreconditionOperator |
                          UnaryConnector PreExpression |
                          PreExpression DualLogicalOperator PreExpression
UnaryConnector ::= "!"
Connector ::= "&&" | "||"
Comparator ::= ">" | "<" | "==" | "<=" | ">=" | "!="
PostconditionSection ::= "<postcondition>" PostconditionExpression "</postcondition>"
PostconditionExpression ::="(" PostconditionExpression")" | PostExpression Comparator PostExpression |
                            UnaryConnector PostconditionExpression |
                            PostconditionExpression Connector PostconditionExpression
PostExpression ::= PostArithmaticExpression | PostLogicalExpression
PostArithmaticExpression ::= "(" PostArithmaticExpression ")" |  PostconditionOperator |
                             UnaryArithmaticOperator PostExpression |
                             PostExpression DualArithmaticOperator PostExpression
PostconditionOperator ::= ParameterName |PostconditionOperator"[" PostArithmaticExpression "]" |
                          "result" | PostconditionOperator "." FieldName| LITERAL
PostLogicalExpression ::= "(" PostLogicalExpression ")" | PostconditionOperator |
                          UnaryConnector PostExpression |
                          PostExpression DualLogicalOperator PostExpression
```

其中,PreArithmeticExpression 和 PostArithmeticExpression 是算术表达式,PreLogicalExpression 和 PostLogicalExpression 是逻辑表达式.UnaryArithmaticOperator 和 UnaryConnector 分别是一元的算术和逻辑运算符,DualArithmaticOperator 和 DualLogicalOperator 分别是二元的算术和逻辑运算符,DualLogicalOperator 包括"与"、"或"和"异或"等.ParameterName 是 Web Services 中方法的输入参数名,它可以是简单类型、数组或复合类型.如果 ParameterName 是复合类型,则 FieldName 是该复合类型的元素名称.result 代表该方法的返回值.LITERAL 可以是 null、数字、字符、字符串或布尔常量等.

我们对 WSDL 语法进行扩展以后,增强了对 Web Services 功能的语义描述,允许服务使用者和服务提供者为 Web Services 对外发布的方法定义多个前置条件和后置条件,这些条件之间是合取关系.Web Services 有其自身特点,服务使用者只能获得它执行完毕后的返回结果,无法获取任何方法的输入参数在执行时的状态.因此,我们扩展的 WSDL 可以使用方法的输入参数的初始值,而无法使用其执行时的值.同样,由于 DbC 中的不变式、循环变式和循环不变式通常需要涉及对象的内部状态,而通过 WSDL 文档无法获取 Web Services 的内部状态信息,因此我们所定义的合约不包括不变式、循环变式和循环不变式等.

通过在 Web Services 的接口描述中加入一定的合约,将有助于分辨服务提供者和使用者的责任.文献[9]提出由服务的使用者在构件的描述信息中加入合约.我们则从服务提供者和服务使用者两个角度来综合考虑 Web Services 的测试,即这两者均可制定合约.

① 服务提供者制定的合约.服务提供者的前置条件描述了使用 Web Services 必须满足的条件,这些条件可以对服务使用者提供一定的指导.服务提供者的后置条件相当于对 Web Services 的部分功能描述.服务提供者的合约可以更全面地说明 Web Services 的功能,从而有助于服务使用者更正确地使用 Web Services.

② 服务使用者制定的合约.服务使用者的前置条件可以保证 Web Services 在应用程序中按照使用者的意图被使用.服务使用者的后置条件用来对 Web Services 的适用性进行测试.服务提供者不可能预想到 Web Services 的所有使用环境和需求,因此最好由服务使用者提出自己的需求.如果 Web Services 中已经包含服务提供者的合约,服务使用者可以直接使用这些合约,或对它们进行修改后使用,也可重新定义自己的合约.

2.2 测试技术框架

基于合约式设计的思想,我们提出了一个 Web Services 的测试技术框架,如图 1 所示.

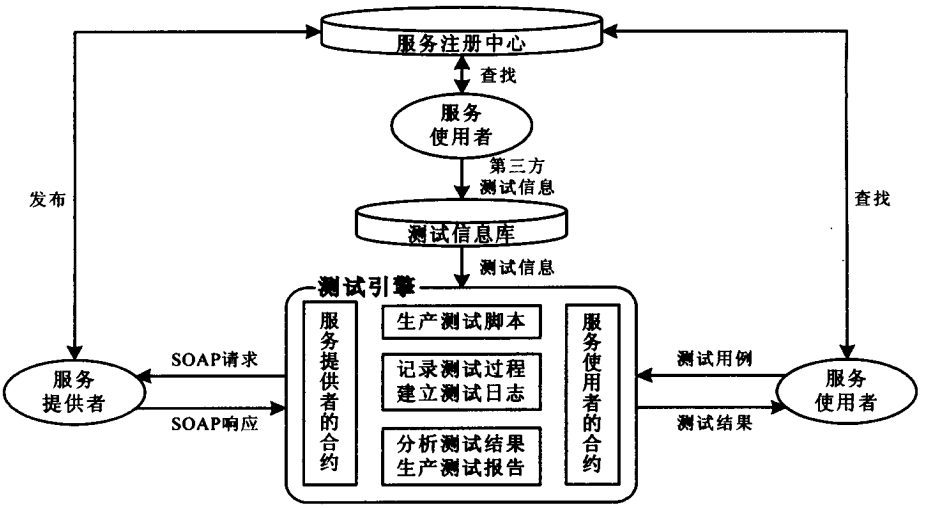

图 1 Web Services 的测试技术框架

服务使用者在服务注册中心查找到所需的 Web Services 后,需要使用 WSDL 文档中的绑定信息,并通过 SOAP 消息来调用 Web Services.服务提供者与服务使用者之间的交互有两种实现方式,一是服务使用者自己书写 SOAP 消息调用服务提供者的 Web Services,并在结果返回后解析 SOAP 消息;二是服务使用者先利用工具(如.Net 中的 wsdl 工具、IBM 的 Web Services Toolkit 等)生成代理类,再通过代理类实现与服务提供者之间的 SOAP 消息传递.我们使用的是第二种方法.服务使用者可以通过测试引擎来调用和测试服务提供者的 Web Services.下面是我们提出的测试引擎的工作流程:

```
Begin
  获得 WSDL 文档;
  添加服务提供者或服务使用者的合约;
  检查合约语法的正确性;
  If (合约语法正确) Then
```

```
    生成含有合约的代理类;
    输入测试用例;
  生成 XML 格式的测试脚本;
  If (测试用例满足前置条件) Then
    Begin
      运行测试用例;
      if (运行结果满足后置条件) Then
        Begin
          比较实际运行结果与期望结果;
          返回比较结果;
          记录比较结果,生成测试报告;
        End
      Else
        返回运行结果不满足后置条件信息,记录并生成测试报告;
    End
  Else
    返回测试用例不满足前置条件信息,记录并生成测试报告;
Else
  返回合约语法错误信息
End.
```

测试引擎可以辅助服务提供者和服务使用者添加 Web Services 的合约,它对 WSDL 文档中的合约进行语法检查.若合约的语法正确,则生成代理类.在程序中使用这些代理类与 Web Services 进行交互.测试引擎接收来自服务使用者的测试用例,然后生成 XML 格式的测试脚本.然后,测试引擎使用待测试 Web Services 的 WSDL 文档中的前置条件来检查该测试用例是否合法.测试引擎将合法的测试用例以 SOAP 消息的形式发送给服务提供者.服务提供者的 Web Services 运行该测试用例后,将 SOAP 消息形式的返回结果发送给测试引擎.测试引擎使用 WSDL 文档中的后置条件检查测试结果,并返回给服务使用者.针对每个测试用例,测试引擎在测试执行时进行合约检查,对于违反前置和后置条件的异常情况,测试引擎都会通知服务使用者.如果运行某个测试用例时,前置条件被违反,说明是服务使用者的使用方式有错;如果后置条件被违反,则说明 Web Services 本身有错.

测试引擎可以对测试结果进行分析,比较该测试用例的实际输出结果和期望结果,将最终的测试结果提交给服务使用者,并生成测试报告.测试引擎记录整个测试过程,建立测试日志,并将测试历史信息存储到测试信息库中,我们可以把这些测试信息视为 Web Services 的反馈信息或附属信息.测试历史信息可以在其他使用者查找 Web Services 时,作为第三方测试信息提供给使用者,有助于其进行 Web Services 的集成测试和回归测试.

3 原型与实验

为了验证本文所提出技术的有效性,我们在 Microsoft .NET 平台上用 C# 语言开发了一个测试 Web Services 的原型(Web services testing tool,简称 WSTT),并且用实际的 Web Services 进行实验.

3.1 实验1

假设服务使用者要实现一个应用,该应用中需要对 5 个整数进行排序并输出其结果,他通过在服务注册中心进行查找,得到某个包含了两个实现冒泡排序的方法的 Web Service,其 WSDL 文档的片断如下:

```
.....
<types>
  .....
  <s:element name="BubbleSort1">
    .....
    <s:element minOccurs="0" maxOccurs="1" name="nums" type="s0:ArrayOfInt"/>
  </s:element>
  <s:element name="BubbleSort1Response">
    .....
    <s:element minOccurs="0" maxOccurs="1" name="BubbleSort1Result" type="s0:ArrayOfInt"/>
  </s:element>
  <s:element name="BubbleSort2">
    .....
    <s:element minOccurs="0" maxOccurs="1" name="nums" type="s0:ArrayOfInt"/>
  </s:element>
  <s:element name="BubbleSort2Response">
    .....
```

```
      <s:element minOccurs="0" maxOccurs="1" name="BubbleSort2Result" type="s0:ArrayOfInt"/>
    </s:element>
  </types>
  <message name="BubbleSort1SoapIn">
    <part name="parameters" element="s0:BubbleSort1" />
  </message>
  <message name="BubbleSort1SoapOut">
    <part name="parameters" element="s0:BubbleSort1Response" />
  </message>
  <message name="BubbleSort2SoapIn">
    <part name="parameters" element="s0:BubbleSort2" />
  </message>
  <message name="BubbleSort2SoapOut">
    <part name="parameters" element="s0:BubbleSort2Response" />
  </message>
  ……
  <operation name="BubbleSort1">
    <input message="s0:BubbleSort1SoapIn" />
    <output message="s0:BubbleSort1SoapOut" />
  </operation>
  <operation name="BubbleSort2">
    <input message="s0:BubbleSort2SoapIn" />
    <output message="s0:BubbleSort2SoapOut" />
  </operation>
  ……
```

从上面 WSDL 文档的片断可以看出,服务使用者能够获得的信息是相当有限的.服务使用者在实现其应用时分别使用了 Web Service 的这两个方法,他测试时所用的一组输入数据为:-1,2,3,-3,5.运行以后他发现,使用方法 BubbleSort1 和 BubbleSort2 后应用的输出结果都是错误的,该测试用例执行失败.

服务提供者的合约有助于服务使用者正确地使用 Web Services,并在使用 Web Services 出错时分辨错误原因.服务提供者可以在该 Web Service 的 WSDL 文档中加入合约如下:

```
  ……
  <portType name="ServiceSoap">
    <operation name="BubbleSort1">
      ……
      <contract>
        <postcondition><![CDATA[nums[0]<=nums[1]]></postcondition>
        <postcondition><![CDATA[nums[1]<=nums[2]]></postcondition>
        <postcondition><![CDATA[nums[2]<=nums[3]]></postcondition>
        <postcondition><![CDATA[nums[3]<=nums[4]]></postcondition>
      </contract>
    </operation>
    <operation name="BubbleSort2">
      ……
      <contract>
        <precondition><![CDATA[(nums[0]>=0) && (nums[1]>=0)]]></precondition>
        <precondition><![CDATA[(nums[2]>=0) && (nums[3]>=0)]]></precondition>
        <precondition><![CDATA[nums[4]>=0]]></precondition>
        <postcondition><![CDATA[nums[0]<=nums[1]]></postcondition>
        <postcondition><![CDATA[nums[1]<=nums[2]]></postcondition>
        <postcondition><![CDATA[nums[2]<=nums[3]]></postcondition>
        <postcondition><![CDATA[nums[3]<=nums[4]]></postcondition>
      </contract>
    </operation>
  </portType>
  ……
```

WSTT 中的测试引擎可以根据如上的合约生成代理类,通过生成的代理类来使用该 Web Service.用与前面同样的测试用例对服务使用者的应用进行测试,发现使用方法 BubbleSort1 时违反了它的后置条件,表明是方法 BubbleSort1 本身功能不正确,属于服务提供者的责任;而方法 BubbleSort2 只对非负数作冒泡排序,使用时违反了方法的前置条件,即使用方式有误,属于服务使用者的责任.

3.2 实验2

假设服务使用者想在自己的应用中使用求一定精度平方根的 Web Service,他在服务注册中心查找后得到某个求平方根的 Web Service 的 WSDL 文档片断如下:

```
.....
<types>
  .....
  <s:element name="ComputeSquareRoot">
    .....
    <s:element minOccurs="1" maxOccurs="1" name="number" type="s:double" />
  </s:element>
  <s:element name="ComputeSquareRootResponse">
    .....
    <s:element minOccurs="1" maxOccurs="1" name="ComputeSquareRootResult" type="s:double"/>
  </s:element>
</types>
<message name="ComputeSquareRootSoapIn">
  <part name="parameters" element="s0:ComputeSquareRoot" />
</message>
<message name="ComputeSquareRootSoapOut">
  <part name="parameters" element="s0:ComputeSquareRootResponse" />
</message>
.....
<operation name="ComputeSquareRoot">
  <input message="s0:ComputeSquareRootSoapIn" />
  <output message="s0:ComputeSquareRootSoapOut" />
</operation>
.....
```

服务使用者仅凭以上的信息很难判断此 Web Service 是否符合自己的精度要求,如果仅靠执行测试用例,通过比较实际输出结果和期望结果来判断 ComputeSquareRoot 的适用性是比较繁琐和耗时的.

服务使用者可以在该 Web Service 的 WSDL 文档中加入如下的合约:

```
.....
<portType name="ServiceSoap">
  <operation name="ComputeSquareRoot">
    .....
    <contract>
      <postcondition>
        <![CDATA[(number-result*result)<0.0001]]>
      </postcondition>
    </contract>
  </operation>
</portType>
.....
```

测试引擎根据测试用例并结合相应的合约进行测试,选取输入数据为 9.8 时,执行结果表明后置条件被违反.通过这种方式,服务使用者可以比较容易地发现该 Web Service 不满足自己的需求.

实验结果表明,使用 WSTT,在对集成了 Web Services 的应用程序进行测试时,如果某个测试用例的实际输出结果与期望结果不符,服务使用者可以很容易地分辨是 Web Services 本身有误还是使用 Web Services 的方法有误,从而对错误进行准确定位,并能判断 Web Services 是否满足自己的需要.因此,本文所提技术是比较有效的.

4 结束语

由服务提供者给出的合约可以为 Web Services 的使用者提供一定的帮助,服务使用者的合约则表达了使用者对 Web Services 的期望.我们的技术定义了这两方面的合约,服务使用者可以利用这些合约来对 Web Services 进行测试,以判定该 Web Services 是否适合应用系统的需求.使用我们的技术可以较好地解决 Web Services 使用中存在的两个问题.我们目前只研究了单个 Web Service 在应用中的测试问题.一旦单个 Web Service 的质量得到保证,就可以使用多个 Web Service 组成的 Web Services flows 来协同完成某项任务.此时,必须对 Web Services flows 进行测试.目前测试 Web Services flows 一般采用模型检验的方法[10, 11],力图在 Web Services flows 实际运行之前检查其可达性或可能出现的死锁等情况.但是,很多问题是在实际运行中才能被发现,因此,我们未来研究的重点是使用合约来对 Web Services flows 进行动态测试.

致谢 本课题的研究得到北京大学信息科学技术学院软件研究所王立福教授的悉心指导,作者在此表示衷心感谢.

References:

[1] Austin D, Barbir A, Garg S. Web services architecture requirements. 2002. http://www.w3.org/TR/2002/WD-wsa-reqs-20020429.

[2] Kreger H.IBM Web services conceptual architecture. 2001. http://www.ibm.com/webservices.

[3] Yang FQ, Mei H, Lu J, Jin Z. Some discussion on the development of software technology. Acta Electronica Sinica, 2002, 30(12A):1901~1906 (in Chinese with English abstract).

[4] Chai XL, Liang YQ. The Technology, Architecture and Application of Web Services. Beijing: Publishing House of Electronics Industry, 2003 (in Chinese).

[5] Tsai WT, Paul R, Wang YM, Fan C, Wang D. Extending WSDL to facilitate web services testing. In: Proc. of the 7th IEEE International Symposium on High Assurance Systems Engineering (HASE 2002), 2002. 171~172.

[6] Tsai WT, Paul R, Song WW, Cao ZB. Coyote: an XML-based framework for Web Services testing. In: Proc. of the 7th IEEE International Symposium on High Assurance Systems Engineering (HASE 2002), 2002. 173~174.

[7] Bertrand M. Object-Oriented Software Construction. Prentice Hall, 1997.

[8] Szyperski C. Components and Architecture. Software Development, 2000.

[9] Mingins CA, Chan CY. Building trust in third-party components using component wrappers in the .NET frameworks. In: The 40th Int'l Conf. on Technology of Object-Oriented Languages and Systems (TOOLS Pacific 2002). 2002. 153~157.

[10] Shin NAKAJIMA. On verifying web service flows. In: Proc. of the 2002 Symp. on Applications and the Internet Workshops (SAINT 2002). 2002. 223~224.

[11] Shin NAKAJIMA. Verification of web service flows with model-checking techniques. In: Proc. of the 1st Int'l Symp. on Cyber Worlds (CW 2002). 2002. 378~385.

附中文参考文献:

[3] 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.

[4] 柴晓路,梁宇奇.Web Services 技术、架构和应用.北京:电子工业出版社,2003.