基于SOA的异构软件自动化测试方法研究*)

李长青　张为群

(西南大学计算机与信息科学学院　重庆 400715)　(重庆市智能软件与软件工程重点实验室　重庆 400715)

摘　要　在异构软件集成领域,采用基于SOA(Services Oriented Architecture)的软件架构及其方法已逐渐成为分布式计算解决方案的主流,但目前针对SOA和Web services的测试方法较少且多以人工测试为主,本文提出构建自动化测试引擎的方法,该方法采用XML语言精化需求规格说明书,自动生成基于XML语言的测试用例,测试引擎自动读取、执行测试用例,记录、分析测试结果,从而可以提高软件开发的效率以及改进软件质量。最后实现了一个自动化测试工具原型,并通过实验的方法验证了本方法的有效性。

关键词　SOA, Web服务, 自动化测试, 异构软件

The Research on Isomerous Software Autotesting Methods Based on SOA

LI Chang-Qing　ZHANG Wei-Qun

(School of Computer and Information Science, South West University, Chongqing 400715)

(The Key Lab for Intelligence Software and Software Engineering, Chongqing 400714)

Abstract　In isomerous software integration domain, Services oriented developing methods are playing more and more important roles in distributed computing resolution. This paper will describe the Web services' traits, and analyze the testing problems of Web services at present. On this condition, we promote a new method based on XML language to solve the problems, and realize a auto testing tool like a test engine to help building test environment, producing test data, executing test operation, recording and analyzing test results. At last, we testify the validation of our method and tool.

Keywords　Web service, Isomerous software, Software auto testing, SOA

1　引言

随着internet的迅猛发展,企业电子商务水平日新月异,以.NET和JAVA为主的两大开发平台及其构造的商业系统在日益激烈的竞争中渐渐成为市场的主角。但是,开发传统的集中式软件所使用的开发平台、开发工具、操作系统在体系结构上的紧耦合性,使得物理分散的独立系统逐渐形成了所谓的"信息孤岛"[6]。为了消除企业内部以及企业之间的信息孤岛,迫切需要一种标准化的、紧耦合的、粗粒度的、开放性好、可集成性高的体系结构来完成计算模式从集中式向分布式转换的重任,于是一种面向服务的体系结构SOA(Service Oriented Architecture)的软件设计方法被提了出来。从核心原理上讲,这种通过XML封装而达到描述标准化、发布标准化、接口标准化、访问标准化的技术可以实现异构软件之间互操作,从而完成异构性质的企业级应用集成[8]。

SOA以其强自治、松耦合、粗粒度、开放性、可集成、互操作、可扩展等特性正逐渐获得了产业界广泛的支持和学术界的重视。同时,作为实现SOA的主流行业标准和参考实现,Web services基于一系列开放的事实标准如SOAP(简单对象访问协议 Simple Object Access Protocol)、WSDL(Web服务描述语言 Web Services Description Language)、UDDI协议(通用描述、发现、集成 Universal Description, Discovery, and Integration)等在异构软件集成领域具有突出的优势:标准的接口、通信协议的规范,极大地降低了系统集成的复杂性和开销,成为企业级异构软件集成解决方案的首选。

由于Web services的强自治、分布性等特性,使得服务的内部结构和设计细节对于服务使用者来说是透明的,况且Web services在运行态被选择、组装、执行,所以基于SOA的异构软件测试极具挑战性。

2　软件测试技术与方法

软件测试是保证服务软件质量和性能的重要技术手段。目前,针对软件领域的自动化测试方法,有静态分析、代码插装、用户事件捕获与回放等方法,但这些方法均属于白盒测试方法,与Web services的黑盒特性是相违背的。

针对Web services的测试方法和工具比较少。姜瑛2005年提出了一种根据WSDL文档中采用随机法自动生成测试数据[6],然后根据合约变异技术选择测试数据,但并未提及如何根据测试数据产生测试用例,以及测试用例的自动执行。郭勇2006年提出了构建Web services测试框架的概念[8],以及如何利用SOA的方法来测试网格软件,但并没有采用有效的方法执行测试用例并分析测试结果。林振提出了基于结构化功能规格说明的测试方法,该方法根据需求规格说明和设计规格说明,生成测试用例以及测试数据集,但该方法对于结构化程序设计比较有效,对面向对象设计以及面向服务的软件不太适合。黄陇2004年提出了构建基于UML sequence diagram模型的自动生成测试用例的方法[7],测试用例的执行通过消息传递机制实现,测试脚本和测试数据采

*)重庆市科委(项目编号 CSTC,2006BA2003)、重庆市信息产业发展基金(项目编号 200611004)。**李长春**　硕士,主要研究领域:软件测试、SOA、Web服务;**张为群**　教授,主要研究领域:形式化软件工程、软件测试。

用关键字驱动技术分离，这样增强了测试脚本的可维护性和可重用性，但完全依靠 UML 工具产生测试数据和测试用例将产生大量的无效测试用例。另外，不根据需求规格说明书产生的测试用例是不符合该软件开发规范的。Tsai（Arizona State University）2002 年提出了构建 Web services 自动化测试框架的想法，该框架配置参数和服务地址，调用 SOAP 协议，执行服务软件，并获取服务软件返回结果进行分析，由此完成对服务软件的自动化测试，但 Tsai 并没有解决存在交互关系的多 Web 服务（即复合 Web services）之间软件测试。由于 Web services 也是某种意义上的构件，所以，采用构件组装关系的描述方法来描述 Web services 是合理的。北京大学杨芙清教授，梅宏教授对构件组装关系早在 1995 年就提出了采用形式化的方法来描述构件及其组装关系。复旦大学任洪敏于 2003 年采用形式化的方法对复合构件进行了描述，重点论述了构件组装关系的机制和行为推导，但缺乏学者将构件组装方法应用到测试领域。

如果有工具自动完成测试用例的调用、执行、记录、分析工作，既可以减轻测试人员的工作量，提高软件开发效率，又可以采用规范的方法提高软件质量，降低软件成本。所以，自动化测试技术是 Web services 发展的关键之一，也是本文研究的核心内容。本文根据 Web services 的特性文档 WSDL 文档和需求规格说明产生测试用例和测试数据，由测试引擎解析测试用例文档，分析各个 Web services 之间的执行关系，从而采用 SOAP 协议调用各个服务，获取服务软件返回结果，测试引擎对测试结果进行统计分析，最终辅助测试人员完成测试工作。

3 自动化测试方法

3.1 自动化测试引擎

目前，大多数 Web services 测试方法均通过配置 Web services 地址、端口以及参数，采用 SOAP 协议来执行单个 Web services 测试用例。W. T. Tsai 提出了建立 Web services 的自动化测试框架，也是采用配置地址、端口、参数，采用 SOAP 协议来执行单个 Web services[2]，这就类似于单元测试。如何针对复合 Web services 的测试，也就是多个交互的 Web services 的测试，目前还有待解决。我们提出了构建自动化测试引擎的方法，它首先将 Web services 的地址、端口和参数配置到 XML 化的测试用例文档中，测试引擎读取测试用例文档，分析该用例内部各个 Web services 之间的复合关系。根据引擎内部核心算法，调用相应的 Web services。获得测试结果后，记录到数据库中，以便于统计与分析。测试引擎的架构如图 1 所示。

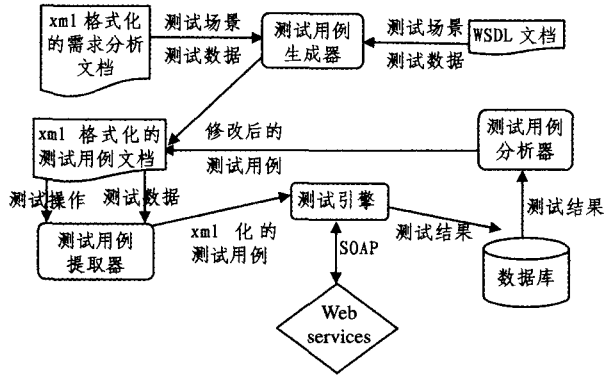

图 1 测试引擎框架图

从图 1 中可以看出，测试数据和测试场景的自动生成是完成自动化测试的前提和基础。对于用户而言，这个软件调用哪个 Web services 并不重要，重要的是该软件是否达到了用户应该满足的功能。而需求分析文档描述了该软件的功能，所以测试用例必须由需求分析文档产生。但需求分析文档采用自然语言描述，具有天生的二义性。首先，转换器辅助将带有二义性的需求分析文档转化成基于 XML 描述的精化的需求文档，其目的在于：①XML 文档描述需求分析更加精细，这有利于减少二义性。②XML 描述的需求规格说明书有利于转换成 XML 描述的测试用例。③XML 需求规格说明书有利于实现与 Web services 的无缝链接。但仅仅根据 XML 描述的需求分析文档生成测试用例场景及数据是不够的，因为 Web services 的动态特性也必须考虑到测试用例中去。恰当的方法是：根据 WSDL 文档与 XML 描述的需求分析说明一起产生该用例的测试数据和测试脚本，既考虑了软件应有的功能，又考虑了服务软件的动态特性。为了更清晰地展现测试引擎自动构建测试场景和获取测试数据的过程，这里有一个跨图书馆书籍查询的 Web service 例子可以清晰展示测试用例的生成过程。我们首先获取 WSDL 文档的信息，如表 1 所示。

表 1 图书查询 Web 服务的 WSDL 文档

```
〈wsdl:types〉
〈complexType name="book"〉〈sequence〉
〈element name="ISBN" type="xsd:string"〉
〈element name="bookName" type="xsd:string"〉〈/sequence〉〈/
  complexType〉〈/wsdl:types〉
〈wsdl:message name="Response"〉〈wsdl:part name="return"
  type="xsd:string"〉〈/wsdl:message〉
〈wsdl:message name="Request"〉〈wsdl:part name="checkBook"
  type="xsd:string"〉〈/wsdl:message〉
〈wsdl:service name="checkBook"〉〈wsdl:port binding=intf re-
  quest〉〈wsdl soap:address〉〈wsdl:address location=" http://
  211.83.30.74/soap/WS_Query1"〉〈/wsdl:port〉〈/wsdl:service〉
〈/wsdl:definitions〉
```

从表 1 中，我们可以获取书籍查询服务的参数接口和服务地址，从而可以据此配置得到测试用例文档。

表 2 配置后的测试用例文档

```
配置后的测试用例文档
    〈scenario name="checkBook"〉
    〈scenario_configuration〉
    〈URN〉BookServices〈/URN〉
    〈SOAProuterURL〉
    http://211.83.30.74/soap/WS_Query1
    〈/SOAProuterURL〉
    〈/scenario_configuration〉
    〈scenario_path〉
    〈method sequence="1"〉
    〈name〉软件工程实践者的方法学〈/name〉
    〈data direction="input"〉
    〈dataname〉isbn〈/dataname〉
    〈type〉string〈/type〉〈/data〉…
    〈data direction="output"〉〈dataname/〉
    〈type〉String〈/type〉〈/data〉〈/method〉
    〈method sequence="2"〉
    〈name〉Bookname〈/name〉
    … 〈/method〉
    〈/scenario_path〉
```

从表 2 中，我们可以清晰地看到测试用例文档是如何从 XML 描述的需求规格文档和 Web services 的描述文档 WSDL 文档中产生。测试引擎将解析测试用例文档，并完成整个自动化测试流程。

3.2 测试用例中 Web services 的执行关系

在测试工具执行复合 Web services 过程中，首先测试用

例文档被解析，测试引擎得到测试文档中各 Web services 之间的执行关系。Web services 之间的基本执行关系分为五种，分别是：顺序执行关系、选择执行关系、分支执行关系、并行执行关系、中断执行关系。其他执行关系可以从这五种关系中组合产生。执行关系采用形式化描述如下。

定义 1 把 Web service 定义为一个四元组〈WSpro，WSsuc，Opt，Qos(WS^i)〉，其中 WSpro 表示 WS$_i$ 的直接前驱集合，WSsuc 表示 WS^i 的直接后继集合。若 WS^i 为第一个起始服务，则 WSpro 为空；若 WS^i 为最后一个服务，则 WSsuc 为空。Opt 表示 WS^i 是否为可选服务，Opt 的取值范围为{0，1}，取值 1 表示选中该服务，否则为未选中。

定义 2 把两个服务之间的互斥关系定义为集合 Alt^i，Alt^i＝{WS$_i$ | WS$_i$∈WS}，该集合为表示该集合中所有服务的属性是多选一的，也就是互斥的。复合 Web services 之间的互斥关系可表示为 Alt，Alt＝{Alt^1，Alt^2，…，Alt^k}。

定义 3 把两个服务之间的依赖关系 Dep^i 定义为一个三元组〈WS^i，WS^j，Attr〉，其中 Attr 的取值范围为{0，1，2，3，4，5}，Attr 为 0 表示无任何依赖关系，Attr 为 1 表示 WS^i，WS^j 属于顺序执行关系，Attr 为 2 表示 WS^i，WS^j 之间属于分支执行关系，Attr 为 3 表示 WS^i，WS^j 之间属于循环执行关系，Attr 为 4 表示 WS^i，WS^j 之间属于并行执行关系，Attr 为 5 表示 WS^i，WS^j 之间属于中断执行关系，复合 Web services 之间的依赖关系集合可表示为 Dep＝{Dep^1，Dep^2，……，Dep^k}。

定义 4 把 Web services 之间的执行关系定义为一个三元组〈WS，Alt，Dep〉，其中 WS 为可能构成该用例的一组 Web services 集合，即 WS＝{WS^1，WS^2，……，WS^n}。

顺序执行关系是指同一测试用例中，各个 Web service 之间的执行关系要么存在严格的依赖关系，要么不存在任何关系。

顺序执行关系如图 2 所示。

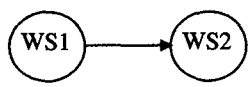

图 2

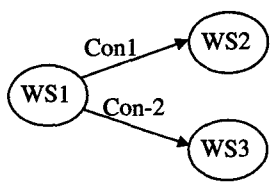

图 3

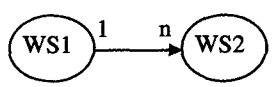

图 4

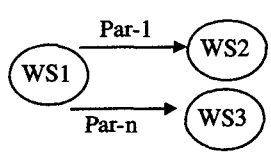

图 5

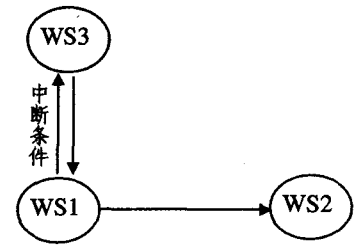

图 6

顺序执行关系是指：执行 WS^1 之后，WS^2 才开始执行，WS^1 与 WS^2 之间存在简单的顺序关系。按照上述定义，顺序执行关系用形式化方法描述如下：

WS＝{WS^1，WS^2}，Alt＝Φ，Dep^1＝{Dep^1}，Dep^1＝〈WS^1，WS^2，1〉

分支执行关系是指同一测试用例中，有两个或多个 Web services 根据某一个 Web service 的执行结果或者某个条件选择执行。这里，我们假设各个分支不存在同步关系。分支执行关系图如图 3 所示。

按照上述定义，分支执行关系用形式化方法描述如下：

WS＝{WS^1，WS^2}，Alt＝Φ，Dep＝{Dep^1，Dep^2}，Dep^1＝〈WS^1，WS^2，2〉，Dep^2＝〈WS^1，WS^3，2〉

根据 WS^1 的结果，选择不同的执行分支。如果结果为 Con1，则执行 WS^2；如果结果为 Con2，则执行 WS^3。

循环执行关系按照上述定义，采用形式化方法描述如下：

WS＝{WS^1，WS^2}，Alt＝Φ，Dep＝{Dep^1}，Dep^1＝〈WS^1，WS^2，3〉

如图 4 所示，循环执行关系中，执行引擎根据 WS^1 的结果处理后再判断，符合某个条件，则循环调用 WS^2，n 次调用后结束。

并行执行关系中，某个 Web services 并行调用多个 Web services。我们假设各个 Web services 之间不存在同步互斥关系。

按照上述定义，并行执行关系用形式化方法描述如下：

WS＝{WS^1，WS^2，WS^3}，Alt＝Φ，Dep＝{Dep^1，Dep^2}，Dep^1＝〈WS^1，WS^2，4〉，Dep^2＝〈WS^1，WS^3，4〉

WS^1 并行调用 WS^2，WS^3。

中断执行关系中，某个 Web service 在调用另一个 Web service 的时候，遇到了突发事件，转去执行中断事件。

WS^1 在调用 WS^2 的过程中，中断条件发生后，测试引擎会转去调用 WS^3。WS^3 的执行对 WS^2 的执行没有影响。

按照上述定义，中断执行关系用形式化方法描述如下：

WS＝{WS^1，WS^2，WS^3}，Alt＝Φ，Dep＝{Dep^1，Dep^2}，Dep^1＝〈WS^1，WS^2，1〉，Dep^2＝〈WS^1，WS^3，5〉

3.3 测试引擎的核心执行算法

```
TestEngine(UsecasePath)
{
    /* 读取 XML 测试用例文档 */
    ReadUsecase(UsecasePath)
    /* 根据测试用例场景 Webservices 结构关系 获取第一个 Web
    service */
    ws=GetFirstWS()
    while(true)
        while(IsTruePro(ws))/* 该 Web service 的前置条件是否具
        备 */
        {
            /* 执行该 Web service 的前置条件 */
            ExecutePro(ws)
        }
        /* 按照 SOAP 协议/HTTP 协议调用该 Web service */
        SOAP_Execute(ws)
```

```
        while(IsTrueSuc(ws))/*该 Web service 的后置条件是否执行完毕*/
        {
            /*执行该 Web service 的后置条件*/
            ExecuteSuc(ws)
        }
        /*根据配置的测试用例文档,循环获取当前 Web 服务的下一个服务*/
        while(wsi=GetNextWS(ws)){
        /*获取 WSsuc 以及与当前 Web service 的执行关系 Relation*/
        Relation=GetRelation(ws,wsi)
        Switch(Relation)
          {
            Case1: Sequence Execution() /*顺序关系*/
            Case2: Devide Execution ()/*分支关系*/
            Case3: While Execution ()/*循环关系*/
            Case4: Parrel Execution()/*并行关系*/
            Case5: Interrupt Execution ()/*中断关系*/
            Default:break;
          }
        }
}
```

3.4 测试结果的记录、分析、统计

测试引擎会在自动完成测试的过程中,记录测试用例的操作步骤与每步相应的操作结果。操作步骤的记录有利于回归测试的执行,匹配测试用例返回格式与期望结果的格式。如果返回格式正确,则将该测试数据放入该测试用例的有效测试数据集;如果返回格式不正确,则将测试数据放入该测试用例的无效测试数据集。如果返回格式正确且返回结果与预期结果不符合,则标记该 Web 服务地址并存入错误库中,以便于测试人员进行手工测试。

4 实验研究

本节将通过测试一个图书查询 Web services 的实际例子来解释自动化测试工具的测试过程。

WS_Query1 采用 C# 语言实现,部署在 IIS 应用服务器上。WS_Query2、WS_Query3 采用 JAVA 语言实现,分别部署在 TOMCAT 和 Weblogic 应用服务器上。自动化测试引擎首先获取 WS_Query1～WS_Query3 的 WSDL 描述文档,再根据 XML 精化的需求分析文档生成测试用例文档,如表 3 所示。

表 3 配置后的测试用例文档例子

```
〈scenario name="checkBook"〉
〈scenario_configuration〉
〈URN〉BookServices〈/URN〉
〈SOAProuterURL〉
http://211.83.30.74/soap/WS_Query1
〈/SOAProuterURL〉
〈/scenario_configuration〉
〈scenario_path〉
〈method sequence="1"〉
〈name〉软件工程实践者的方法学〈/name〉
〈data direction="input"〉
〈dataname〉isbn〈/dataname〉
〈type〉string〈/type〉〈/data〉…
〈data direction="output"〉〈dataname/〉
〈type〉String〈/type〉〈/data〉〈/method〉
〈method sequence="2"〉
〈name〉Bookname〈/name〉
… 〈/method〉
〈/scenario_path〉
```

测试引擎执行过程如下:

(1)测试引擎读取测试用例文档,获取图书查询的 Web 服务地址和发送参数(书名);

(2)测试引擎模拟服务的请求者向 Web 服务 WS_Query1 发送书名查询该图书是否存在;

(3)若返回格式错误,预期结果为(书名、ISBN 号、简介、价格),如果返回结果与此不符合,首先将这个 Web 服务地址标记为无效地址,再查找备选的图书查询 Web 服务地址列表,选择其他的 Web 服务地址,重新发送请求再次查询,开始新一轮的等待;

(4)若返回格式正确,但返回的 ISBN 号为空,可以认为该书不存在;

(5)若返回格式正确,可以读取 ISBN 号、简介、价格存入临时库中;

(6)测试引擎循环调用备选库中其他图书查询 Web 服务地址列表,选择其他 Web 服务 WS_Query2—WS_Query3 地址,调用这些 Web 服务可以获取所有的查询结果,返回格式错误的 Web 服务标记为无效地址,并将这些结果存入临时库中;

(7)测试引擎调用对比分析 Web 服务 WS_fx,若返回格式正确,得到推荐的购买书名以及 ISBN 号、价格;若返回格式不正确,将 WS_fx 标记为无效地址。

自动化测试引擎生成的测试记录如表 4。

表 4 测试记录示例

```
……
测试时间:2006-12-22 14:28
测试服务:WS_Query1
测试标记:失效
测试内容:测试数据库中记录为空时,返回结果是否为空。
测试预期结果:返回字符串为"该书不存在,请检查书名是否有误?"
测试结果:返回格式错误
对软件功能的分析:该功能不符合预期结果
对该功能的建议:暂无
……
```

测试引擎触发测试用例分析器生成的缺陷分布图和功能状态图如图 7 和图 8。

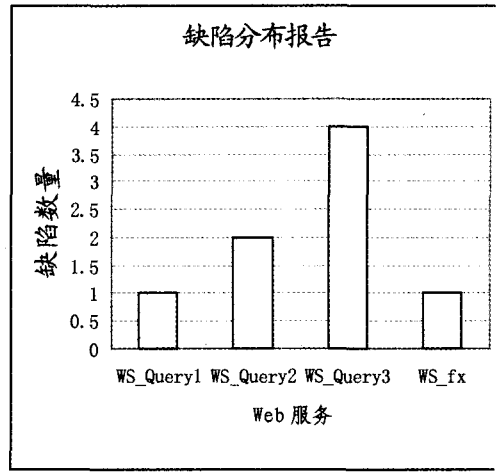

图 7 缺陷分布图

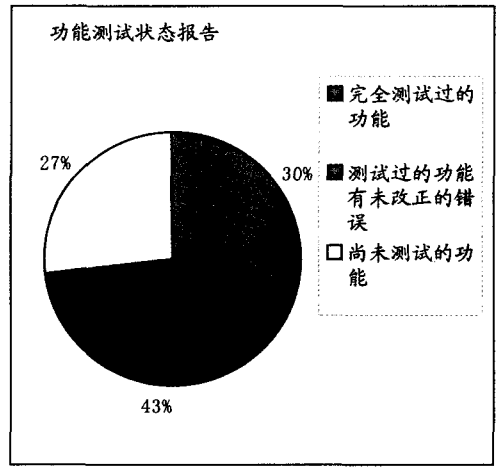

图 8 功能测试状态图

实验结果表明：该测试引擎能够尽快发现错误，定位错误位置，分析错误类型，提高测试覆盖率和测试精度。

结论和未来研究方向 本文针对基于SOA的Web services软件的特点，并充分利用测试领域的相关知识和方法，在此基础上提出了基于XML技术的自动化测试引擎的核心算法，该算法的关键在于对复合Web services之间关系进行了模型描述。提出了顺序关系、分支关系、选择关系、并行关系等五种基本关系，通过配置测试用例，自动提取测试数据，采用SOAP协议对Web services进行访问并记录、分析测试结果，最终完成Web services集成测试。该方法对Web services软件集成有较大的帮助。

随着分布式计算的深入，Internet在应用广度的发展，使得智能设备、手机、PDA等设备使用Web服务的范围逐渐扩大，对Web服务软件采用自动化测试方法已是大势所趋。本文所提出的测试方法还有一些不足，如何针对移动设备及WAP协议完成服务软件自动化测试将是下一步研究的工作。

参考文献

1 Bai Xiaoying, Dong Wenli, Tsai W T, et al. WSDL-based Automatic Test Case Generation for Web Services Testing. In: Proc. of the 2005 IEEE International Workshop on Service-oriented System Engineering (SOSE' 05), 2005

2 Tsai W T, Paul R, Song W, et al. Coyote: An XML-based Framework for Web Services Testing. In: Proc. of the 7th IEEE International Symposium on High Assurance Systems Engineering (HASE'02), 2002

3 Tsai W T, Paul R, Wang Y, et al. Extending WSDL to Facilitate Web Services Testing. In: Proc. of IEEE HASE, 2002. 171～172

4 Tsai W T, Chen Y, Cao Z, et al. Testing Web Services Using Progressive Group Testing. In: Proc. of the Advanced Workshop on Content Computing, 2004. 314～322

5 Tsai W T, Chen Yinong, Paul R, et al. Adaptive Testing, Oracle Generation, and Test Case Ranking for WebServices. In: Proc. of the 29th Annual International Computer Software and Applications Conference (COMPSAC), Edinburgh, July 2005. 101～106

6 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. 计算机学报，2005，28(4)

7 黄陇，于洪敏，陈致明，等. 基于UML的软件测试自动化研究. 计算机应用，2004，24(7)

8 郭勇，邓波，衣双辉. 面向服务的网格软件测试环境. 软件学报，2006，17(11)

9 赵俊峰，谢冰，张路，等. 一种支持领域特性的Web服务组装方法. 计算机学报，2005，28(4)

(上接第267页)

两种方法，用SPN描述每次抗衰的具体实施过程，用DFA控制各次抗衰的返回位置，明确描述各抗衰子过程之间的关联与转移。这样既避免了模型状态空间爆炸问题又避免了采用DFA模型不能描述策略的实施细节的缺点。由此，由此得出了简约明确的系统嵌套的进程级重启抗衰策略实施模型，并以其易于理解和分析的优点，为全面实施软件系统智能化细粒度软件抗衰策略提供支持。

参考文献

1 Garg S, Moorsel A V, Vaidyanathan K. A Methodology for Detection and Estimation of Software Aging. In: Proceedings of the 9th International Symposium on Software Reliability Engineering, Paderborn, Germany, 1998

2 Huang Y, Kintala C, Kolettis N. Software Rejuvenation: Analysis, Module and Applications. In: Proc. of FTCS-25, Pasadena, CA, 1995

3 Castelli V, Harper R E, Heidelberger P. Proactive Management of Software Aging. IBM JRD, 2001, 45(2): 311～332

4 Hong Y, Chen D, Li L. Closed Loop Design for Software Rejuvenation. In: Workshop on Self-Healing, Adaptive and self-MANaged Systems (SHAMAN), New York, 2002

5 Xiea W, Hongb Y, Trivedi K. Analysis of a two-level software rejuvenation policy. Reliability Engineering and System Safety, 2005, 87(1): 13～22

6 Patterson D, Brown A, Broadwell P. Recovery Oriented Computing (ROC): Motivation, Definition, Techniques, and Case Studies. [Technical Report]. UCB/CSD-02-1175. UC Berkeley Computer Science 2002

7 Candea G, Fox A. Recursive Restartability: Turning the Reboot Sledgehammer into a Scalpel. In: 8th Workshop on Hot Topics in Operating Systems, Schloss Elmau, Germany, 2001

8 Candea G, Kawamoto S, Fujiki Y. Microreboot - A Technique for Cheap Recovery. In: 6th Symposium on Operating Systems Design and Implementation, San Francisco, CA, 2004

9 Candea G, Cutler J, Fox A. Improving Availability with Recursive Microreboots: A Soft-State System Case Study. Performance Evaluation Journal, 2004, 56(1-3): 213～248

10 王湛，游静，赵颜利，等. 基于访问关系的进程重启相关性判定. 计算机科学[J]，2006，33(9)

11 游静，徐建，张琨. 计算系统软件抗衰重启技术研究[J]. 信息与控制(已录用，待发表)

12 You Jing, Xu Jian, Zhao Xue-long, et al. Modeling and Cost Analysis of Nested Software Rejuvenation Policy. LNCS 3612, 2005. 1280～1289

13 You Jing, Xu Jian, Zhao Xue-long, 等. Modeling and Availability Analysis of Nested Software Rejuvenation Policy. In: IEEE SMC, 2005

14 Murthy C S R, Manimaran G. Resource Management in Real-time System and Network. Cambridge, MA: MIT Press, 2001