一种基于改进的贝叶斯投票算法的服务评估技术*)

张燕生[1] 白晓颖[2] 蒋长征[2]
(中国科学院研究生院 北京 100049)[1] (清华大学计算机系 北京 100084)[2]

摘 要 现有的Web服务体系架构缺少服务质量与优选方面的有效支持,如何动态地选择、绑定并调用最适合用户需求的Web服务受到研究领域的关注。本文基于对"可信服务测试中介"模型的研究,提出了一种基于改进的贝叶斯投票算法的服务评估技术——把每一个测试评估结果作为对测试对象的一次投票,通过对协同服务测试数据的收集和分析,最终计算出同类服务的加权通过率,并在同类服务中进行排行,体现了Web2.0的技术特点。

关键词 Web服务,测试中介,协同测试,Web2.0,服务排行

A Technique for Evaluating Services Based on Improved Bayesian Voting Algorithm

ZHANG Yan-Sheng[1] BAI Xiao-Ying[2] JIANG Chang-Zheng[2]
(Graduate University of Chinese Academy of Sciences, Beijing 100049)[1]
(Department of Computer Science and Technology, Tsinghua University, Beijing 100084)[2]

Abstract Web services architecture lacks of effective support for quality of service and Selection. It is an ongoing research direction in Web services community how to dynamically select, bind, and invoke Web service which can best meet the requirements of service consumer. Researching in the model of "Trustworthy Service Test Broker", it is proposed one technique for evaluating services based on improved bayesian voting algorithm on this paper. Every test result from collaborative test is thought as one pool for the test target. Through collecting, analyzing these collaborative test data, the pass rate of every service can be calculated, and then services of same type can be ranked. The technique has the characters of Web2.0.

Keywords Web services, Collaborative test, Web 2.0, Services rank

1 引言

作为一种新型的计算模型,面向服务架构(Service-Oriented Architecture, SOA)以及它的主要实现方式——Web服务(Web Services, WS)正在重新定义软件开发的过程。这为整个IT业注入了新的机遇,同时也带来更多的挑战,需要产生新的技术来支持这种变革。

Web服务是一种松散耦合的、基于规约的、模块化的、崭新的分布式计算模型。它通过XML、SOAP、WSDL、UDDI、WSFL、BPEL等开放协议和标准,提供面向互联网应用的统一服务注册、发现、绑定、及集成调用机制。基于标准的动态协同是Web服务的一个主要特点。UDDI模型在服务提供方(Service Provider)、服务中介(Service Broker)和服务请求方(Service Requester)之间为服务的发布、发现和绑定提供了一种解决方案。然而SOA发展的瓶颈是——用户很难在众多的相同定义的服务中找到真正符合实际需求的服务。正如布隆博格早在2002年所指出的那样:"Web服务的成功与否取决于在其测试方面的解决能力"[1],在"软件即为服务"的今天,Web服务作为一种软件的表现形式与传统软件一样需要进行严格的测试,只是所采用的标准、技术和方法不同。

首先,在服务质量评估和定义方面,需要建立新的服务质量评估和定义的标准,使参与服务的三方能够有依可循,为此,UDDI在v3版本中加入了与安全和质量相关的属性。另外在文[2]中的服务评估模型考虑采用了与服务领域无关的服务属性作为服务评价因子,其AgFlow系统中选择以下5个通用的服务属性来评价Web服务:服务运行成本(execution price)、运行时间(execution duration)、信任度(reputation)、成功率(successful execution rate)和可用性(availability)。评价者根据自身经验设定服务属性的权重,评价结果为这5个评价因子的取值加权后的总和。因为这种评价模型所选择的服务属性与服务领域无关,所以可以用来评价任何类型的Web服务;在文[3]中提出了一种领域自适应的服务评估模型,其目的是为了克服前一种模型所忽略的与服务领域相关的服务属性的质量要求的缺点。

虽然在服务质量标准方面,有以上这些有益的尝试,但如果只建立了服务评估和定义的标准,而其评估测试过程仅仅是由服务提供者或中介进行,那么其评估结果的可信性相对较低。因为由Web服务自描述的任何属性,无论是功能性的还是非功能性的,如果没有经过严格的协同测试和验证,都不可避免地带有参与服务的某一方的倾向性,缺乏可信的基础。

因此在服务质量评估的可信性方面,需要建立一个可信的服务测试评价中介(Trustworthy Testing & Evaluating Service Broker, TTSB)为参与服务的各方提供一个跨平台的、分布式的、多代理的协同测试。这样,由于服务测试环境各异、服务测试资产和测试评估的结果来自于参与服务测试的各方,因此测试评估的结果更具有可信性。本文的研究就

*)项目资助:国家自然科学基金项目(编号:60603035)。张燕生 高级工程师,硕士研究生,研究方向为软件测试;白晓颖 博士,副教授,硕士生导师,研究方向为软件测试;蒋长征 本科生。

是基于这样一个协同服务测试评估中介模型所进行的，为协同测试系统研究了一种基于协同测试评估数据的、具备Web2.0特点的、改进的贝叶斯投票算法用于服务质量排行。以下将就此算法的研究背景、原理、改进以及排行系统的实现等方面进行论述。

2 研究背景

在传统的SOA应用中，UDDI所扮演的只是一个服务中介(Service Broker)的角色，它只是为需要发布服务的服务提供方和需要发现服务的服务请求方提供一个目录服务，并不能承担起服务质量保证的责任，服务提供者一经数字签名或其它方式的验证身份后，就可以在传统的UDDI服务器上注册所要发布的服务，这些服务所提供的服务质量(包括性能、可靠性和效能等因素)都无从保证。而文[4]中提出的可信的服务测试中介则是把传统的UDDI服务器扩展成具有协同服务测试能力服务中介(Test Broker)的模型，它提供一种发布和发现测试资源的能力，使所有的测试资产拥有者可以自由地交流和协同测试与验证。

协同测试与验证(Collaborative Validate & Verification，CV&V)是把Web2.0中"用户参与"和"数据为核心"的理念应用到了软件测试领域，利用大众的资源和智慧使Web服务的质量得到测试和验证。服务质量的评估是由大众参与测试的结果所产生的，而且参与评估的用户越多、测试次数越多、测试纬度越广，其评估的结果就越可信。同时，CV&V系统还为用户提供了一个测试资产共享复用的平台。

这种测试中介(Test Broker)服务器可以被分布在各地、针对不同的问题承担不同的协同测试验证任务，同时还能彼此协作，每一个服务在"check in"或"check out"或是任何需要的时候，都会被测试和评估，实时地为用户提供可信的、具有质量保证的服务。Test Broker模型可以使参与Web服务测试的多方进行协同测试和验证。测试提供者可以提供诸如测试用例、可执行的测试脚本、测试结果和缺陷等资产；测试者可以利用这些发布的测试资产在实际的测试环境下对服务进行测试；系统可以基于测试统计的数据自动地进行测试排行、测试者信誉排行、服务提供者信誉排行和服务排行等工作。在这个系统中，测试者和测试提供者可以是参与服务三方中的任意一方：

1. 测试者可以是服务提供者，他要对他所提供的服务质量负责；也可以是服务请求者，他要用测试用例来检验所选择的服务；还可以是服务中介，它要确保每一个注册的服务都能提供优良的服务质量。可以是某个组织，它在某个领域内提供专业的服务质量认证；也可以是任何一个独立的测试者，他热衷于为公众提供服务质量评估信息。

2. 测试提供者可以是服务提供者，他连同服务一起发布相关的测试说明；可以是服务请求者，他要发布符合他所要求的测试需求的测试资产；也可以是某个独立的测试者，他为某些发布的服务提供相应的测试资产。

这种可信的服务测试中介模型如图1所示。

其中各方所进行的协同测试活动描述如下：

1. 测试提供者(Test Provider)通过测试中介(Test Broker)发布测试资产(Test Artifacts)

2. 测试者检查测试中介并搜索测试用例；

3. 测试者得到测试用例、脚本或与测试提供者的链接。

4. 如果测试用例不能被测试者独立执行，可以与测试提供者建立协同测试。

5. 测试中介维护测试用例与服务之间的映射关系。

6. 测试者从服务中介(Service Broker)中找到与某测试用例相关的服务。

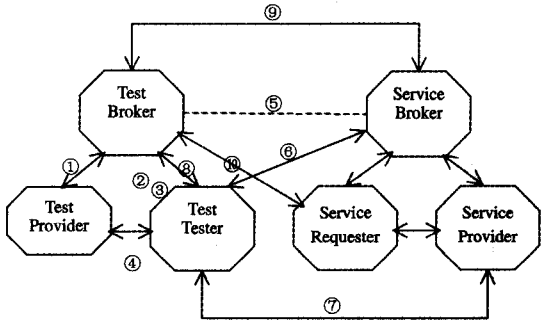


图1 可信的服务测试中介模型

7. 测试者绑定服务提供者提供的服务，并且对此服务进行测试、获得测试结果。

8. 测试者向测试中介发布测试结果和缺陷报告，以便进行进一步的服务评估。

9. 测试的过程可以集成到服务发布的过程中。服务中介可以和测试中介一同在"Check-in"和"Check-out"或任何需要的时候进行协同测试，以便只允许拥有满意质量的服务才能发布给用户调用。

10. 测试过程也可以集成到服务发现的过程中。服务请求方(Service User)可以把测试中介提供的测试结果和评估作为服务选择的参考。

在可信的服务质量评测中介的研究方面有许多具体实现问题需要解决，如在这种可信的服务中介的架构上，如何为参与测试的各方搭建一个实用的、有效的、B/S模式的、协同测试和验证的平台，使用户可以在此平台上发布Web服务、测试用例以及测试执行结果，也可以查看所有已发布的Web服务、测试用例以及测试执行结果与评价，还可以通过平台管理用户发布的测试资源，并综合测试执行结果自动地为Web服务进行实时的、动态的排行，本文重点阐述的是在获得协同测试统计数据后，如何利用改进的贝叶斯投票算法对服务进行排行。

3 服务排行算法

在收集了大量的服务测试结果数据之后，对相同类型的服务进行服务质量排行是一个主要需要解决的问题，因为排行算法的优劣直接影响到协同测试系统的性能和质量。为提高服务质量排行的可信度，我们采用了改进的贝叶斯投票算法。

3.1 贝叶斯投票算法

贝叶斯投票算法[5]成功地应用在世界上最大的互联网电影数据库(IMDB)的排名系统中，算法简单、运行效率高、结果准确、抗干扰能力强。此算法源自于民主投票的思想，在这里，把服务排行看作一种投票行为，如果某一服务的测试执行结果是"通过测试"，就相当于测试者对此服务投了一个赞成票，反之则投了反对票。每个服务的测试通过率高相应地排行应该靠前。但是如果只是简单地根据测试通过率进行服务排名，会出现这样一个不合理的现象：

"如果某个服务基于100次测试的通过率是80%，另一个服务只经过一次测试但通过率是100%，结果后者的排名

却在前!”

因此为了解决以上这种排名的不真实性,采用加权均值的办法:如果某个服务的测试次数较少,那么这些测试就不如测试次数多的服务的那些测试更可信。这就意味着测试次数多的服务的测试结果具有更高的“可信度”;如果某服务经过越多的测试,它真实的服务质量就越靠近它实际被测试的通过率;如果某服务经过越少次测试,它真实的服务质量就越靠近此类服务的平均通过率。这种算法就叫“贝叶斯投票算法”或“贝叶斯排名算法”,具体的计算公式如下:

$$WR(s)=\frac{(\frac{1}{n}\sum_{i=1}^{n}N_i)\times(\frac{1}{n}\sum_{i=1}^{n}R_i)+N_s\times R_s}{\frac{1}{n}\sum_{i=1}^{n}N_i\times N_s} \quad (1)$$

$WR(s)$ 是服务“s”的加权通过率,N 是某个服务的总的测试次数。R 是某个服务的通过率。n 是此类服务的服务个数。式(1)可简化成:

$$WR(s)=\frac{(\frac{1}{n}\sum_{i=1}^{n}P_i)+P_s}{\frac{1}{n}\sum_{i=1}^{n}N_i+N_s} \quad (2)$$

这里,P 是服务的通过次数,其它如式(1)所述。

3.2 改进的算法

上面的排行算法把所有的测试者置于都同一个可信度水平上,认为所有的测试者发布的测试结果都对服务质量评估发挥同等效力的作用。但现实情况是:由于每个测试者所处的地位、所拥有资源和经验不同,当他们对同一个服务进行测试时,所采用的测试用例、测试针对性、测试的覆盖率都有可能不同,这样,即使得到相同的测试结果——“通过”或“不通过”,这些测试结果对服务质量评估的准确度是不同的,因此可信度也是不同的。一般来讲,专业测试机构所做出的测试结果比业余测试爱好者的测试结果可信度更高,更能代表服务的实际质量水平;另外,既使是同一个测试者,随着他参与测试次数的增加,经验的积累,其测试水平、测试的可信度也会越来越高。

因此,本系统把测试者分成两类:“专业测试者”和“业余测试者”,并如表 1 所示为不同的类型的测试者设置不同的初始信誉度权值,该权值会随着测试者发布测试次数的增加而不断增加。

表 1 信誉度权值的设置

	信誉度权值 k	
测试次数	专业测试者	业余测试者
1～10	11～20	1～10
11～20	21～30	11～20
21～30	31～40	21～30
…	…	…

服务排行算法可以被改进为带有测试者信誉度权值的计算公式,如下所示:

$$WR(s)=\frac{(\frac{1}{n}\sum_{\substack{i=1\sim n\\ j=1\sim m}}k_jP_{(i,j)})+\sum_{j=1\sim m}k_jP_{(s,j)}}{\frac{1}{n}\sum_{\substack{i=1\sim n\\ j=1\sim m}}k_jN_{(i,j)}+\sum_{j=1\sim m}k_jN_{(s,j)}} \quad (3)$$

这里,k 代表某个测试者的权值,m 是所有测试者的个数,其它如式(2)所述。

从上式可以看出,每个参与服务测试的测试者的信誉度权值会直接影响到其测试结果的可信度,从而影响该服务的质量排名结果。

另外,为了能够得到较为真实同类服务的平均通过率,本系统采用以下两个策略来限制取样范围:1. 设定用户单位时间内发布测试的次数,以便剔除那些为了拉动排行恶意频繁发布测试结果的行为;2. 设定最低的信誉度权值,只拥有高于此权值的用户所发布的测试结果才会被选入采样范围。通过这些措施,系统可以自动地、有效地抑制不良干扰,提高排名的可信度。

为实时、动态地实现服务排行,每当用户提交了一个新的测试资源时,平台就立即更新相关的测试者的信誉度权值、同类服务的平均加权通过率和该服务的加权通过率,并刷新排行。由于每次计算都是在上次更新数据的基础上迭代进行的,因此排行计算速度很快。

3.3 算法模拟实验分析

在系统中,测试者的信誉度权值是动态变化的,其测试结果的可信度、与被测服务同类的服务加权平均通过率也是随之变化的,最终影响到被测服务的排名也是动态变化的。因此平台在每一次发布的测试时做出响应,动态地、迭代地计算出服务排行。

图 2,3,4 是由 Matlab 模拟数据进行的算法实验产生的数据对比图。数据模拟了一组同类服务(S1, S2, ……)的测试结果数据,在 S2 等同类的多个服务发布了几百次数据之后,S1 服务连续发布了 300 次测试数据,这三幅图主要对比显示的是 S1 的实际测试通过率、贝叶斯加权通过率及改进的贝叶斯加权通过率的变化规律。

图 2 显示的是 S1 服务的实际测试通过率曲线。可以看出,在初期由于测试次数较少,统计的测试通过率曲线的随机波动较大,大约在发布 50 次左右测试后,曲线趋于平稳至 70%左右的通过率。

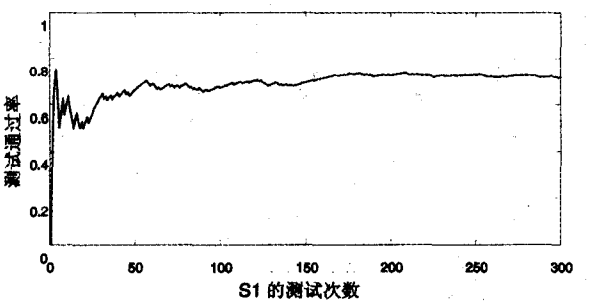


图 2 S1 实际的测试通过率的变化曲线

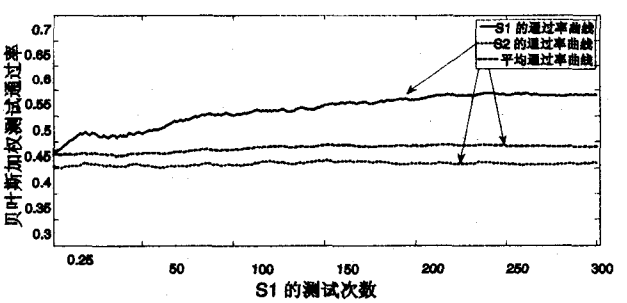


图 3 贝叶斯加权通过率的变化曲线

图 3 显示的是 S1、S2 服务的贝叶斯加权通过率曲线以及同类服务的平均通过率曲线。可以看出 S1 服务的加权通过率在初期由于测试次数较少,受到同类服务平均加权通过

率的严重影响，趋近平均值，与图 1 的曲线相比，抑制了通过率的随机波动；而在后期，随着发布测试次数的增加逐渐趋近于实际通过率。S2 服务由于在此段没有发布测试，因此曲线基本没有变化。

图 4 显示的是由不同信誉度权值的测试者对同一个的服务 S1 发布相同的测试结果的改进的贝叶斯加权通过率，从图中看出，测试者的信誉度越高，其测试结果对通过率计算值的影响就愈大，只需要发布较少的测试结果就可以把曲线拉向其测试通过率的附近。

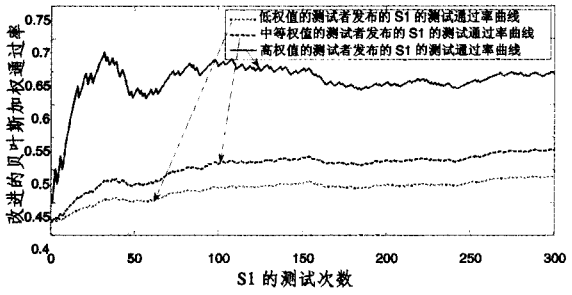


图 4　改进的贝叶斯加权通过率的变化曲线

4　服务质量排行的实现

平台主体采用 J2EE 技术结合 Struts 和 Hibernate 框架的设计方案。Struts 架构解决了视图层、业务层和控制层的分离；Hibernate 架构则提供了灵活的持久层支持；通过将这两个架构整合起来，可以得到一个迅速地开发灵活、低耦合及易于维护的信息系统的完整解决方案[6]。Struts 负责降低系统总架构的耦合性，而让 Hibernate 负责降低业务模型部分的开发难度。平台的架构设计图 5 所示。

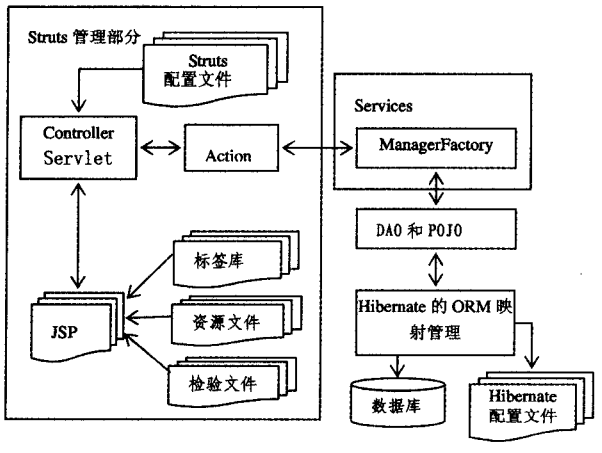


图 5　平台整体框架图

在线的动态排名，受所有用户操作的影响，在平台启动时要根据以前平台的数据存储初始化排行，当用户提交操作后平台可以实时更新排行。所以必须在 application 域内建立空间来组织维护排名所需要的相关信息。

为此本平台中建立了以下几个类，其类图如图 6。

ServiceCreditInfo：此类的一个对象对应一个服务，属性为每个服务的 id、服务的测试次数，通过次数，测试执行总权值，以及通过的测试执行的权值以及评估结果。

ServiceTypeCrediInfo：此类的一个对象对应一个服务类别，属性为每个服务类别的 id，服务类别内测试执行权值之和，通过的测试执行权值之和，一个 Hashtable：services，里面的元素是实例化的 ServiceCreditInfo 类，是本服务类别内所有服务对应的 ServiceCreditInfo 实例，一个 ArrayList，ArrayList 内的元素都是 Hashtable 中元素的引用。

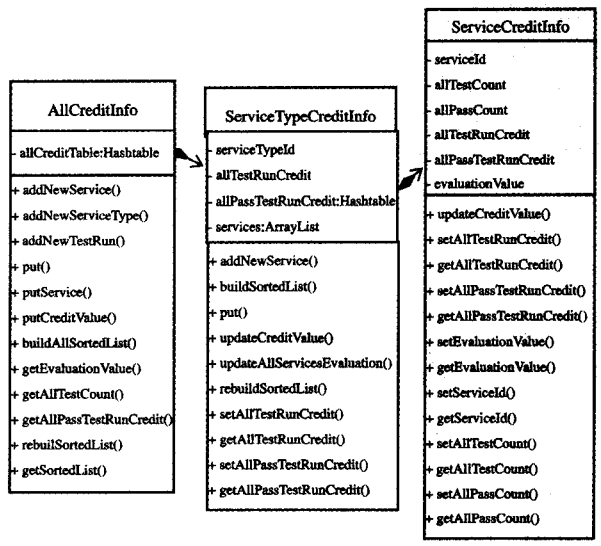


图 6　排行算法实现中的类图

AllCreditInfo：其属性定义了一个 Hashtable，里面的元素是实例化的 ServiceTypeCreditInfo，是本平台所有服务类别对应的 ServiceTypeCreidtInfo 实例。

算法的数据结构图如图 7。

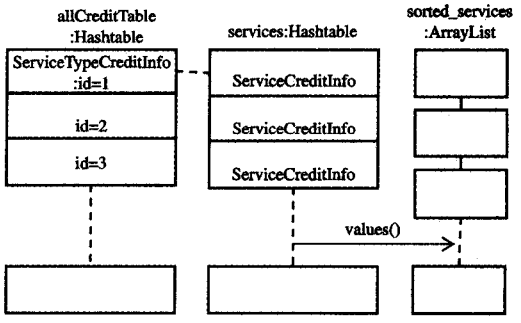


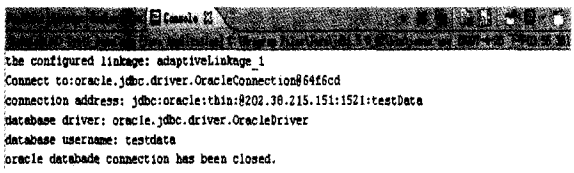
图 7　算法数据结构图

要实现在线的动态排行首先必须进行初始化：在 ExtendActionServlet 的 init()方法中实例化 AllCreditInfo 类的一个对象 allCreditInfo，调用 ServiceTypeManager 的 getAll()方法得到所有的服务种类，根据每个服务种类生成 ServciceTypeCredit 的实例化，并调用 AllCreditInfo 中的 put 方法添加到 allCreidtTable 中。调用 ServiceManager 的 getAll()方法得到某一服务类别的所有的服务，实例化每个服务，并添加到所属服务类别 ServiceTypeCredit 对象的 services 中。调用 TestRunManager 的 getAll()方法得到目前平台已有的所有测试执行，将测试执行的权值以及是否通过的信息累加到测试执行所属服务中去。所有服务类别中的 ArrayList 元素通过 Hashtable 的 values()方法得到所有元素的引用，并调用 Collections 的静态方法 sort()执行排序。sort 方法需要在方法的参数中指定一个比较器，程序中自定义了一个比较器 ServiceCreditComparator。

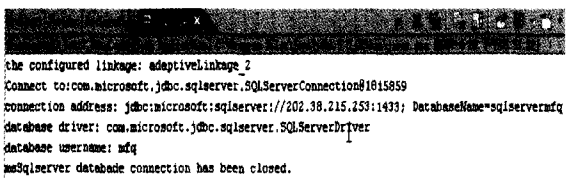
其次是每当发生以下动作发生时，更新信息并重新对

（下转第 272 页）

动态行为重构过程中应提供的功能和各功能组件间的关联关系，同时设计给出各功能组件间的交互逻辑关系模型，然后基于动态代理机制并扩展引入委托主体接口类和统一接口抽象类，实现动态行为重构件功能模型以及功能组件间的交互逻辑模型，并用形式 java 语言详细描述出实现过程中的主要类和关键交互逻辑的设计方式。最后我们设计一需求案例场景进行实例开发实验，通过实验检验出本文所设计给出的构件动态行为重构功能模型以及交互逻辑模型是正确和可行的，通过动态行为重构实现功能重构是有效的。从而为设计实现适应性构件提供一种新途径。



a 运行结果



b 运行结果

图 5

参 考 文 献

1 Fowler M, Beck K, Brant J, et al. Refactoring: Improving the design of existing code[M]. Addison Wesley Publishing Company, 1999

2 What is Refactoring? [EB/OL]. http://www.refactoring.com

3 Wake W C. Refactoring Workbook[M]. Addison Welsley, 2003

4 Mao FQ, Qi DY, Lin WW. SC: An approach to ensure software reliability and reusability[C]. In: Proceedings of the 5th International Conference on Distributed Computing and Applications for Business, Engineering and Sciences, DCABES 2006 PROCEEDINGS, VOLS 1AND 2, 2006. 1244～1249

5 Mao Feiqiao, Qi Deyu, Liao Qiliang, et al. Adaptive Linkage: an Interface Level Adaptable Component Development Technique [C]. In: Proceedings of he 6th International Conference on Control and Automation(ICCA'07), May 2007(to be published)

6 Dynmic Proxy Classes[EB/OL]. http://java.sun.com/j2se/1.3/docs/guide/reflection/proxy.html

7 Eclipse sdk3.2[EB/OL]. http://www.eclipse.org/downloads/

8 Xerces-J—src.2.9.0.zip[EB/OL]. http://archive.apache.org/dist/xml/xerces-j/

9 Log4j v1.2.9[EB/OL]. http://logging.apache.org/log4j/docs/download.html

10 Peng Xin, Wu Yijian, Zhao Wenyun. A Feature-Oriented Adaptive Component Model for Dynamic Evolution[C]. In: Software Maintenance and Reengineering, 2007. CSMR '07. 11th European Conference onMarch 2007. 49～57

11 Zewdie B, Carlson C R. Adaptive Component Paradigm for Highly Configurable Business Components[C]. In: Electro/information Technology, 2006 IEEE International Conference on, May 2006. 185～190

12 Jeong Ah Kim, Jin Young Taek, Sun Myung Hwang. Rule-based component development[C]. In: Software Engineering Research, Management and Applications, 2005. Third ACIS International Conference on, Aug. 2005. 70～74

13 Szydlo T, Szymacha R, Zielinski K. Policy-based Context-aware Adaptable Software Components for Mobility Computing[C]. In: Enterprise Distributed Object Computing Conference. EDOC'06. 10th IEEE International, Oct. 2006. 483～487

14 Bastide G, Seriai A, Oussalah M. Dynamic Adaptation of Software Component Structures[C]. Information Reuse and Integration, 2006 IEEE International Conference on, Sept. 2006. 404～409

(上接第 258 页)

ArrayList 元素排序：

1. 添加服务种类：当新增一个服务种类后，需新初始化一个 ServiceTypeCreditInfo 对象，并添加到 allCreditInfo 的哈希表中。

2. 添加服务：新提交一个服务后，需初始化一个 ServiceCreditInfo 对象，并添加到服务所属类别的哈希表以及 ArrayList 中。

3. 添加测试执行：新提交一个测试执行后，需将测试执行的结果累加到所测试服务的 ServiceCreditInfo 中，并调用 sort()方法。

图 8 为系统实现服务排行的效果图，在浏览器中，参与测试活动的所有用户可以自由地发布服务、测试用例、以及测试用例中的测试步骤，也可以对以上信息进行浏览，并实时地得到某类服务动态的排行情况。

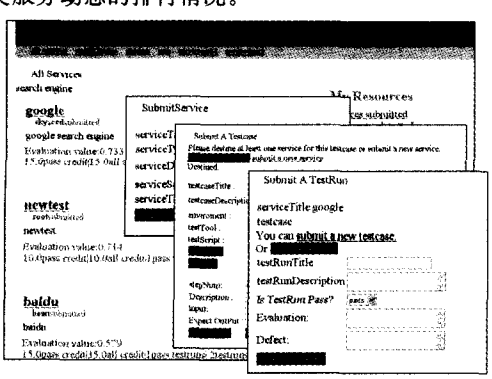


图 8 系统实现效果图

结束语 通过对"可信服务测试中介"模型的研究，提出的这种基于改进的贝叶斯投票算法的服务评估技术是为测试服务的各方搭建了一个共享与重用测试资产的协同测试的平台，并综合分析协同测试数据、实时地为同类服务进行动态排行，由于采用了带有用户信誉度加权平均的服务质量评估算法，使系统具备了用户交互和数据综合的 Web2.0 的功能，在完全信任测试信息发布的真实性的理想情况下，其排行的可信度会随着测试者、测试次数的增多而提高。系统要进一步实用化，还可以在服务排行算法中综合考虑多方面多种因素的影响，如测试者可信度、测试用例可信度、测试覆盖率等，另外还需要完善测试安全认证、多代理控制、智能检索等方面的功能，并且系统本身也可以封装成服务的形式被再组装或调用。

参 考 文 献

1 Bloomberg J. Web Services Testing: Beyond SOAP, ZapThink LLC. http://www.zapthink.com (2002)

2 Bai X, Cao Z, Chen Y. Design of a Trustowrthy Service Broker and Dependence Based Progressive Group Testing. In: International Journal of Simulation and Process Modeling(IJSPM), 2006

3 McCall J A, Richards P K, Walters G F. Factors in Software Quality. RADC(Rome Air Development Cent re): [Technical Report RADC2TR2772363]. 1977

4 杨文军，李涓子，王克宏. 领域自适应的 Web 服务评价模型. 计算机学报，2005，28(4)

5 Allen C, Appelcline S. Collective Choice: Rating Systems' http://www.lifewithalacrity.com/2005/12/collective_choi.html

6 冯国仕，李志蜀. 基于 Struts 与 Hibernate 集成架构的项目管理系统. 计算机应用，2005，l25(8)：1884～1889