适应性构件设计实现关键问题研究*)

毛斐巧　齐德昱
(华南理工大学计算机科学与工程学院　广州 510640)

摘　要　适应性构件应具备灵活应对功能需求变化的能力。为使构件支持这一特性，本文研究设计给出构件行为动态重构功能模型及其交互逻辑模型，并基于动态代理机制实现该模型，通过重构构件的行为实现构件执行功能的转换，从而使构件可灵活适应功能需求的变化。最后的实例开发实验证实了这种设计实现方式的可行性以及行为重构的有效性。

关键词　适应性，构件，重构，动态代理

Research on Key Problems in Adaptive Component Design and Implementation

MAO Fei-Qiao　QI De-Yu
(College of Computer Science & Engineering, South China Univ. of Tech., Guangzhou 510640)

Abstract　Adaptive component should be able to flexibly adapt to function requirement changes. In order to support this characteristic we design and provide a component behavior restructuring function model and an interaction logic model, then we implement the models based on dynamic proxy mechanism, and so we implement function transition for a component by dynamically restructuring its behavior and enable the component to flexibly adapt to function requirement changes. At last, an example development experiment shows that the designed models are effect and the implementation approach is feasible and effective.

Keywords　Adaptability, Component, Restructuring, Dynamic proxy

1　引言

欲使构件能够灵活应对功能需求的变化，通过重构构件功能是一种解决途径，关键是如何实现构件功能的动态重构。在软件开发技术领域内，传统的重构(refactoring)概念是在不改变软件行为的前提下改变软件的代码结构，去掉代码中的“坏味道”，使软件易理解和维护，或者应用一系列重构技术重新实现软件，重构前后功能不发生改变[1~3]，是一种代码级的重构。区别于传统概念，这里的重构(restructuring)是重新构造构件功能，功能发生改变，是一种功能级的重构。实现功能级的重构是比较困难的。针对该关键问题，本文通过动态重构构件的行为，改变构件执行功能的动作，从而达到功能重构的目的。在我们之前研究提出的适应性构件概念模型[4]以及适应链接开发技术[5]的基础上，进一步研究设计给出构件行为动态重构的功能模型(文中第 2.1 节)，在 2.2 节以序列图的方式给出功能模型中各功能组件间的交互逻辑模型，紧接着在 2.3 节基于动态代理机制实现功能模型中的各组件以及它们之间的交互逻辑，详细给出关键部分的实现方式，最后在第 3 节描述实例开发实验的过程和结果。

2　行为动态重构设计实现

2.1　重构功能模型

适应性构件行为重构功能模型主要描述构件中参与行为重构所需要的各组件以及组件间的关联关系。

如图 1 所示，构件中参与行为重构的各功能组件是虚线框内的模型元素，虚线框表示是构件内部。这里说明各模型元素及其代表的组件在行为重构中所负的功能。

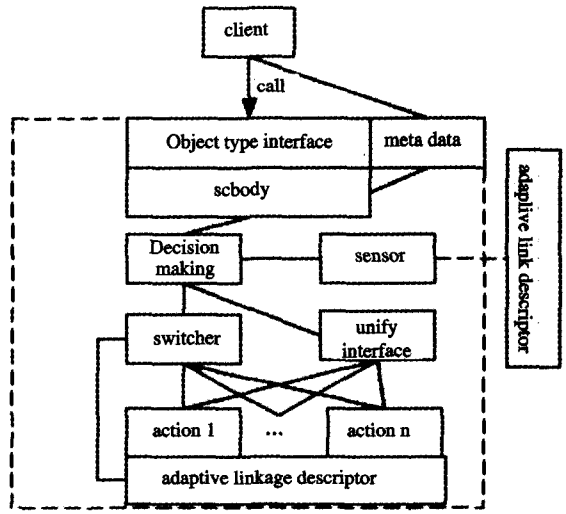


图 1　行为重构功能模型

对象类型接口(object type interface)元素：代表对象类型接口组件，该组件负责向构件外部提供对构件以对象调用方式访问的接口功能，与构件的服务类型接口相对应。

构件主体(scbody)元素：代表构件主体组件，该组件负责启动决策器组件，并使用决策器给出的底层动作访问接口与对象接口配合实现构件对外提供的功能的执行动作。

*)粤港澳关键领域重点突破项目(2005A10307007)、广东省自然科学基金(05300200)、广东省重大科技专项(2003A1030404)。**毛斐巧**　博士研究生，主要研究方向为软件工程、分布式系统；**齐德昱**　教授、博士生导师，主要研究方向为软件工程、分布式系统、网络安全、网格计算等。

元数据(meta data)元素:代表元数据组件,该组件负责描述构件执行动作的操作对象(即构件的属性),并向构件外部提供数据入口。

决策器(decision-making)元素:代表决策器组件,是适应性构件概念模型的主要组成部分[4],该组件在这里主要根据感知器组件给出的信息负责决策出构件所需动态加载的动作组件。

感知器(sensor)元素:代表感知器组件,也是适应性构件概念模型的主要组成部分[],该组件在这里主要负责感知构件外部适应链描述符(适应链描述符是一种适应性业务逻辑配置描述组件,其中描述有构件的适应链接配置信息,我们将另有文章描述)中自己所属构件的适应链接[5]配置信息。

转换器(switcher)元素:代表转换器组件,该组件负责执行动态重构行为动作,根据适应链接描述符组件描述的动作组件的配置信息,动态加载动作组件,实现构件行为的动态转换。

统一接口(unify interface)元素:代表统一接口组件,该组件负责为各动作组件提供模板接口,并向决策器提供统一形式的底层动作访问接口。

动作 i(action $i, i \in N$)元素:代表各动作组件,该类组件负责实现执行不同行为的动作。

适应链接描述符(adaptive linkage descriptor)元素:代表适应链接描述符组件,该组件负责描述与各动作组件相对应的适应链接配置信息,在文[5]中我们有对适应链接描述符和适应链接规范的详细描述。

另外,图中的各实连接线元素表示组件间的功能对象引用关系,虚连接线表示构件内部组件与外部组件间的参考关联关系,客户(client)元素代表访问构件的客户构件,与构件行为重构没有直接功能作用,只是为了辅助描述。

2.2 交互逻辑模型

适应性构件行为重构交互逻辑模型主要描述功能模型中各组件随时间发生的交互逻辑关系,这里用 UML 序列图描述。

如图 2 所示,按时间顺序,客户构件开始访问前先调用元数据组件创建元数据组件对象;元数据组件返回其对象 dataobj 给客户构件;客户构件同步调用对象类型接口组件和构件主体组件开始创建主体组件对象;主体组件创建并调用决策器组件对象;决策器创建并调用感知器件对象,感知器对象感知适应链组件中构件的适应链接配置信息,并将感知结果 alinkageinfo 返回给决策器;决策器决策出要加载的动作组件 action$i$ 后创建调用转换器组件对象进行动作转换;转换器按照适应链接描述符组件中动作组件 action$i$ 的配置信息同步创建统一接口组件对象和创建并加载动作组件 action$i$;统一接口组件将其对象 UIobj 返回给转换器;转换器组件再将其返回给决策器;决策器再将其返回给主体组件;然后主体组件将对象 scobj 返回给客户组件。通过这一系列交互,构件的行为重构完成,客户构件开始使用获得的 scobj 对象访问构件。

当客户构件的访问构件的功能需求发生变化,构件需要重构其行为进行功能转换时,先更改适应链描述符组件中的构件适应链接描述配置,然后重新经过上面的一系列交互,客户构件获得的 scobj 对象会提供与之前不同的功能,已经发生了行为重构。

2.3 实现方式

动态代理是 java 中动态加载类的一种程序设计机制[6]。为实现上述的行为重构功能模型以及功能模型中组件间的交互逻辑关系模型,我们借鉴动态代理机制,将转换器设计实现为动态代理主体类,动作组件设计实现为委托主体类。同时对该机制进行扩展,引入委托主体接口类实现动作 i 组件接口,引入统一接口抽象类实现统一接口组件。下面用形式 java 语言描述实现中用到的主要类和关键交互逻辑的设计,及实现中其它重要问题的解决。

(1)Isc name 类:该类实现对象接口组件,其设计方式如下。

```
public interface Iscname{
    {public returnType opname ( paraType paraname , … );
    …}
```

(2) scname MetaData 类:该类实现元数据组件,其设计方式如下。

```
public class scname MetaData
    {public attributeType attributename ;…
    public scname MetaData ( ) {
    //set initial value for attribute defined above.
    … }
```

(3)Iscname Subject 类:该类实现统一接口组件,其设计方式如下。

```
abstract public interface Iscname Subject {
    abstract public returnType opname
    ( paraType paraname ,… );
    … }
```

(4)Sensor 类:该类实现感知器组件,其设计方式如下。

```
pubic class Sensor {…
    //感知适应链接配置信息
    public int senseLinkage ( ) { … return linkageFlag;//返回适应链接配置信息给决策器 }
    … }
```

(5)Iactioniname 类:该类是我们在代理机制基础上引入的委托主体接口类,实现动作 i 组件的访问接口,其设计如下。

```
public interface Iactioniname {
    public returnType opname ( paraType paraname , … );
    … }
```

(6) actioninameImpl 类:该类是委托主体类,实现动作 i 组件,其设计方式如下。

```
public class actioninameImpl implements Iactioniname , Isc name Subject {
    private attributeType attributename ;
    …
    public returnType opname ( paraType paraname , … );
    … }
```

(7)DynamicSubjectscname 类:该类是动态代理主体类,实现转换器组件,其设计方式如下。

```
public class DynamicSubjectscname implements Invocation-Handler {
    private Object sub;
    public DynamicSubjectsc name ( ) { };
    private DynamicSubjectsc name ( Object obj ) {sub = obj; }
```

```
public Object invoke ( Object proxy, Method method, Object [] args) throws Throwable {
    Object obj = method. invoke ( sub, args );
    return obj; }
//创建统一接口组件对象以及创建并加载动作组件 i
    public static Object createBuilder ( Object toSub ) {
//将统一接口组件对象返回给决策器组件
    return Proxy. newProxyInstance (
    tosub. getClass(). getClassLoader(), tosub. getClass(). getInterfaces, new DynamicSubjectscname(tosub)); }
```

(8) Decisionmaking 类:该类实现决策器组件,这里我们使用简单的 i = = action$i$ 策略规则进行决策,其设计方式如下。

```
public class Decisionmaking { …
    public IscnameSubject decisionClass ( ) {
    Sensor sensor = new Sensor();//创建感知器组件对象
    int opt = sensor. senseLinkage();//调用感知器对象,获得适应链接配置。
    IscnameSubject p = null;
//根据策略进行决策。
    swithch (opt) {
    case i :
```

//创建调用转换器组件对象进行动作转换,转换器获得统一接口组件返回的对象。

```
    p = (IscnameSubject) DyanamicSubjectscname . createBuilder(new actioniImpl());
    break;
    default: break; }
    return p;//将统一接口组件对象返回给主体组件。}
    … }
```

(9) scnameImpl 类:实现构件主体组件,其设计方式如下。

```
public class scnameImpl implements Iscname { …
    IscnameSubject subject = null;
    publicscnameImpl () { …
//创建并调用决策器组件对象。
    Decisionmaking decisionmaking = new Decisionmaking ();
    subject = decisionmaking. decisionClass();
    … }
    public returnType opname ( paraType parname , … ) {
    return subject. opname ( paraname ,… );
    或 subject. opname ( paraname?, …); return; }
    … }
```

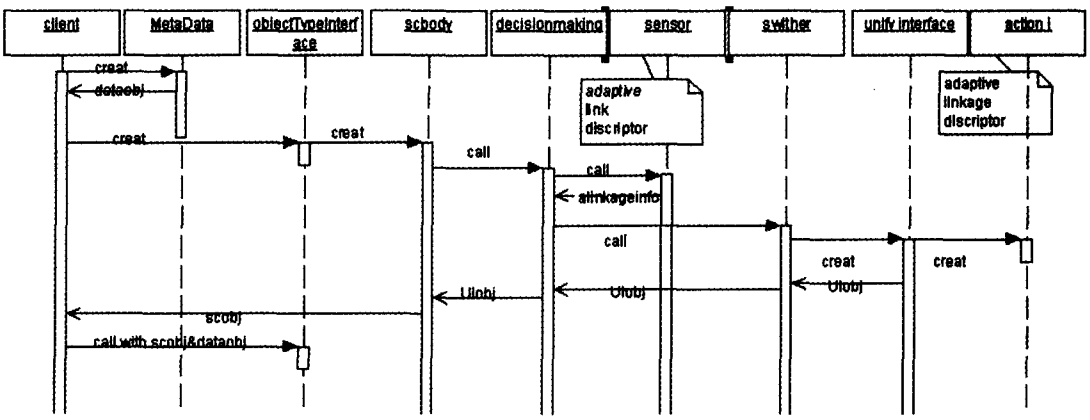


图 2 行为重构交互逻辑模型

另外,客户构件中关键交互点部分设计方式为:

```
scnameMetaData metadata = new scnameMetaData ();//创建并获得返回的元数据组件对象。
//同步调用对象类型接口组件和创建并获得返回的主体构件对象。
    Iscname scname = new scnameImpl();
//使用获得的主体构件对象访问构件。
    scname. opname (metadata, …);
    或 returnType var = scname. opname (metadata, …);
```

而适应链接符组件和适应链组件均实现为 xml 格式配置文件,适应链接描述符的实现规范在文[5]已有描述,对于适应链的设计实现我们将另有文章描述。

3 实例开发实验与结果

为检验所设计的行为重构功能模型、交互逻辑模型和实现方法的正确性、可行性,通过行为重构实现功能重构的有效性,我们设计以下需求案例场景,进行开发实验。

需求案例场景设计:开发一个数据库连接构件,提供数据库连接功能,能灵活适应两种不同类型数据库(Oracle 型数据库和 Microsoft SQL server 型数据库)连接。

开发实验过程中使用的软件环境主要是在免费开源可扩展开发平台 Eclipse sdk v3. 2[7] 中扩展集成免费开源解析器 Xerces-Jv2. 9[8](主要在感知器实现类中用于解析适应链描述符配置内容),扩展集成免费开源日志工具 Log4j v1. 2. 9[9](主要用于输出信息),扩展集成 Oracle 数据库驱动库 classes12. jar 以及 Microsoft SQLserver 数据库驱动库 mssqlserver. jar。

使用的硬件环境主要是用于开发的主机(IP:202. 38. 244. 218),运行 Oracle 数据库服务的主机(IP:202. 38. 215. 151,数据库实例:testData,连接帐户:testdata),运行 SQL server 数据库服务的主机(IP:202. 38. 215. 253,数据库:sqlservermfq,连接帐户:mfq)。

开发实验中我们将此构件命名为 DataConn,客户端命名为 Test1_DataConn。为此构件设计了两个行为组件 OracleConn 行为组件和 SqlDataConn 行为组件,分别执行连接 Oracle 数据库和 Sql 数据库行为。实现该实例的详细设计如

图 3 所示(用 UML 类图描述)。

其中,AdaptiveLink. xml 中对此构件的配置为:

〈 scomponent scid="scdataConn" … 〉

〈adaptiveLinkage〉 adaptiveLinkage_1

〈/adaptiveLinkage〉

…〈 /scomponent 〉

其中,Decisionmaking 的决策策略为 1==OracleConnImpl,2==SqlDataConnImpl。

其中,实现访问此构件的客户构件类 Test1_DataConn. java 代码内容如图 4 所示。第 7 行创建并获得 DataConn 构件的元数据组件对象;第 8 行创建并获得 DataConn 构件的实例对象 connsc;第 9 行使用对象 connsc 访问构件获得数据库连接;第 10 行使用对象 connsc 访问构件获得连接地址信息;第 12 行使用对象 connsc 访问构件获得所连接的数据库的驱动信息;第 14 行使用对象 connsc 访问构件获得连接数据库所使用的帐号;第 16 行使用对象 connsc 访问构件关闭连接。

当执行图 4 中的客户构件时,控制台(因为我们 log4j 输出目标配置为控制台)的输出结果如图 5(a)所示。而将 AdaptiveLink. xml 中此构件的适应链接配置为 adaptiveLinkage_2 时,执行图 4 中的客户构件,控制台输出结果如图 5(b)所示。比较 a 和 b 结果,显然 DataConn 构件运行行为发生了重构,实现了需求案例场景中期望的功能转变。

通过该实例开发实验,我们检验出所设计的构件行为重构件功能模型和交互逻辑模型是正确和可行的,并且通过行为动态重构实现功能重构也是行之有效的。

4 相关研究工作

文[10]通过为构件指定基于特征的行为协议控制构件的行为,使构件的行为能随具体业务的变化而变化,这种行为控制是比较细粒度的,实现构件具体行为的操作方法要按照指定的协议执行,正如研究者说举的例子:一个订购方法的行为控制协议规则在库存量大于 100 与小于等于 100 两种条件下是不同的。Zewdie B,等[11] 基于适应性构件范例(ACP,Adaptive Component Paradigm)提出一种层次模式适应性构件设计方式并给出参考结构。Jeong Ah Kim 等[12] 提出一种基于规则的适应性构件设计方法,主要采用将构件的相对稳定的基本属性与构易变属性分离方式设计构件,以提高构件的功能适应性。Szydlo T,等[13] 通过提出一种基于策略的框架为其适应性构件的适应性提供支持,其适应性构件称为自发对象(SO,Spontaneous Object),其适应性机制主要依赖于为 SO 创建运行时环境的容器。与我们的研究相似,Bastide G,等[14] 也是关注构件的重构,但在其研究所提出的方法中,重构前后构件的行为不发生变化,重构的目的是为了改善构件的结构。

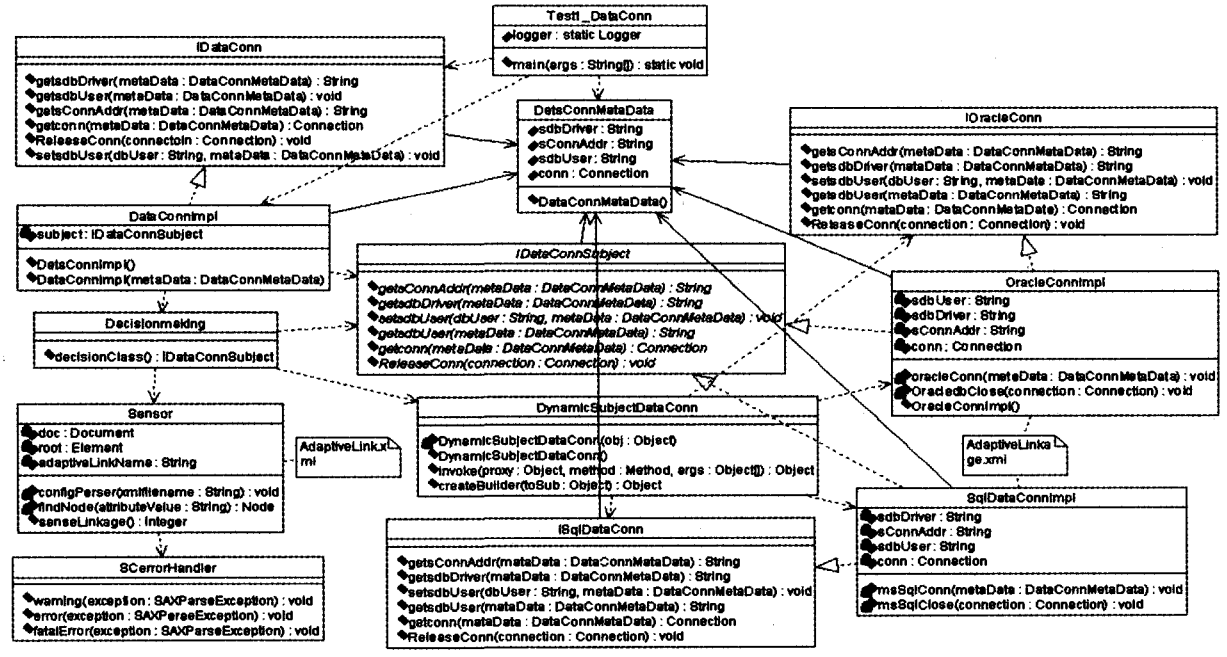


图 3 案例实现详细设计

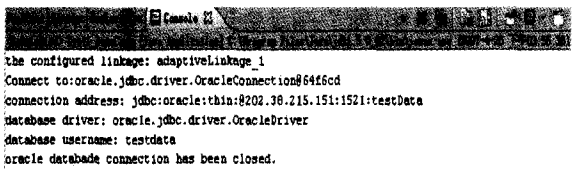
```
1  package sf.dataServiceLayer.scdataConn;
2* import java.sql.Connection;
3  import org.apache.log4j.Logger;
4  public class Test1_DataConn {
5      public  static Logger logger = Logger.getLogger(Test1_DataConn.class);
6*     public static void main(String[] args ) {
7          DataConnMetaData metaData = new DataConnMetaData();
8          IDataConn connsc = new DataConnImpl();
9          Connection conn = connsc.getconn(metaData);
10         String str = connsc.getsConnAddr(metaData);
11         logger.info("connection address: "+str);
12         str = connsc.getsdbDriver(metaData);
13         logger.info("database driver: "+str);
14         str = connsc.getsdbUser(metaData);
15         logger.info("database username: "+str);
16         connsc.ReleaseConn(conn);
17     }
18 }
```

图 4 客户构件实现

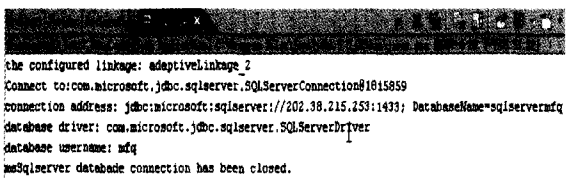
虽然我们的研究工作也是针对如何使构件具备适应能力,但区别于上述研究,我们研究构件的适应性与业务功能需求相关联,强调通过较大粒度的动态行为重构,进行构件执行功能的动态转换,从而使构件能够灵活适应变化的功能需求。我们着重研究如何进行构件的动态行为重构,给出具体可行的行为重构设计实现方式,为设计实现适应性构件提供一种新途径。

结论 本文针对如何设计实现适应性构件功能重构关键问题,提出通过动态行为重构方法来实现。在我们之前研究给出的适应性构件概念模型以及适应链接开发技术的基础上,设计给出构件动态行为重构功能模型,指定各功能组件在

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。

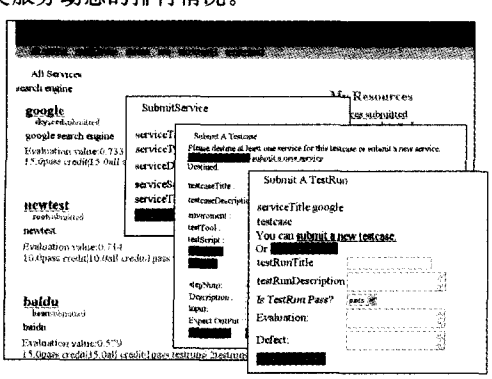


图 8 系统实现效果图

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