一种基于运行时体系结构的 BPEL 支撑环境

马 骞, 虞建杰, 马晓星, 吕 建
(南京大学计算机软件新技术国家重点实验室, 南京大学计算机软件研究所, 江苏南京 210093)

摘 要: 为了使 BPEL 描述的组合服务能够根据动态的网络环境和多变的用户需求而动态演化, 在我们已有工作的基础上, 提出一种基于运行时体系结构的 BPEL 支撑环境. 其核心在于引入一个运行时体系结构对象来刻画组合服务的体系结构, 并用其解耦组合服务与其成员之间的引用关系, 从而通过对该对象的修改引起组合服务与其成员之间交互行为的重解释, 实现组合服务的动态演化. 在此支撑环境上开发了一个简单的应用实例以展示动态调整的效果.

关键词: 面向服务的计算; 服务组合; BPEL; 软件体系结构; 动态演化
中图分类号: TP311 文献标识码: A 文章编号: 0372-2112 (2006) 12A-2360-06

A Runtime Software Architecture-enhanced BPEL Supporting System

MA Qian, YU Jiar-jie, MA Xiao-xing, LÜ Jian
(*State Key Laboratory for Novel Software Technology, Institute of Computer Software, Nanjing University, Nanjing, Jiangsu 210093, China*)

Abstract: An enhanced BPEL supporting system is proposed based on our previous work to facilitate the dynamic adaptation of composite services, which is often desirable and sometimes unavoidable to keep the services facing evolving environment and varying user requirements satisfactory. With this approach the runtime architecture object which describes the architecture of a composite service is utilized to further decouple the composite service from its constituents by redefining the service references under the current architectural context. The dynamic adaptation is implemented by reinterpreting communicating behaviors between the composite service and its constituents, which is naturally resulted from the modification of the runtime architecture object. A simple application is also developed with the supporting system to illustrate the effect of this architecture-based dynamic adaptation mechanism.

Key words: service-oriented computing; services composition; BPEL; software architecture; dynamic evolution

1 引言

面向服务的计算(Service-Oriented Computing, SOC)逐渐成为 Internet 开放环境下构建分布式应用和电子商务解决方案的范型(Paradigm)[1]. Web Service 为 SOC 的实现提供了一条可行的技术途径. 它的价值不仅在于使不同组织开发的软件实体能够以服务的形式通过开放标准(WSDL, UDDI, SOAP)在 Internet 上加以描述, 发布和使用, 更为重要的是可以将已有的 Web Service 有效的组织起来完成复杂的业务功能, 因此如何组合 Web Service 以提供更具价值的"组合服务"成为目前研究的一个重点. 近年来, 服务组合语言大量涌现, 其中 BPEL4WS(简写为 BPEL)[2] 逐渐成为事实上的标准. 它对业务流程中各个 Web Service 的交互行为建模, 以有序的方式协调它们之间的交互活动达成特定的应用目标[3].

然而, 人们逐渐意识到, BPEL 并不能解决 Internet 开放环境下服务组合面临的所有问题. 虽然在组合服务设计阶段可以通过有效的查询机制寻找合适的成员服务, 但是目前 BPEL 组合服务与各个成员服务之间却保持着一种静态的绑定关系, 从而导致在运行期难以动态调整其结构, 当环境发生变化时, 组合关系只能重新建立和描述[4]. 而作为基于 Internet 的面向服务的计算系统, 对如何适应动态的网络环境和多变的用户需求常有很高的要求, 由于局部的改变而导致组合服务整体的重新部署通常是无法接受的.

我们在前期工作[5]中给出了一种基于运行时体系结构的面向服务的动态协同架构, 说明了运行时体系结构在组合服务动态演化中的应用, 但是其与主流的 BPEL 服务组合方式并不兼容. 本文针对上述问题, 提出一种基于运行时体系结构的 BPEL 支撑环境. 旨在为 BPEL 组合服务引入一个运行时体

收稿日期: 2006-09-22; 修回日期: 2006-11-10
基金项目: 国家自然科学基金(No. 60403014); 国家 973 重点基础研究发展规划(No. 2002CB312002); 国家 863 高技术研究发展计划(No. 2006AA01Z159); 江苏省自然科学基金(No. BK2006712)

系结构对象来解耦其与成员之间的绑定关系, 从而能够根据体系结构的当前配置解释组合服务与其成员之间的交互行为. 同时, 对体系结构对象的修改会造成交互行为的重解释, 实现 BPEL 组合服务的动态演化. 本文的工作一方面扩展了原有架构, 为其引入 BPEL 描述组合服务的业务逻辑, 另一方面使该架构与 BPEL 执行引擎相结合, 可以为 BPEL 组合服务的开发, 部署, 运行和演化提供有效的支持. 我们通过在该架构的支撑平台 ARTEMIS-ARC 中开发一个实际场景下的组合服务实例来展示本文工作的可行性和有效性.

2　BPEL 服务组合方式分析

BPEL 是一种基于工作流的服务组合语言, 把组合服务刻画为一个能够实现特定功能的业务流程. BPEL 流程是通过一系列的基本活动和结构化活动, 以及它们的执行顺序来表达的. 任何可执行的 BPEL 流程可以通过 BPEL 执行引擎加以解释执行. 与组合服务发生交互的各个成员被称为"伙伴(partner)", 通常组合服务在部署前需要已知它们的 WSDL 描述, 从而能够根据它们定义的接口类型(PortType)在< invoke> 活动中表达对指定接口的调用关系. 在运行期组合服务会根据 WSDL 描述中成员服务的访问点向其发送请求消息和接收响应消息. BPEL 中组合服务与其"伙伴"的交互行为可以归结为:

partner. portType(params);

这种交互行为的一个显著特点是在调用关系发生前 partner 必须获得一个具体的值(通常为成员 Web Service 的 URL), 也就是说在组合服务部署前, 其对某种接口的请求必须与实现该接口的成员服务加以显式的静态绑定. 我们认为, 虽然这并不阻碍业务功能的实现, 但是它基于两个假设: 一是所绑定的成员服务必须始终是可用的; 二是所绑定的成员服务必须始终是最符合用户需求的.

然而, 处于开放环境中的服务组合通常是由跨越企业组织的多个 Web Service 间相互协作实现的, 从而网络环境的改变, 成员服务的技术升级, 新服务的出现, 旧服务的消亡以及非功能性需求的变化等等情况都可能造成以上两点假设不同程度的破坏. 针对这种变化, BPEL 服务组合方式现有的解决方法是停止原服务的运行, 进行重新开发和部署. 但是, 组合服务采用这种方式以应对环境的变化往往会带来无法接受的延迟, 高额的代价和巨大的风险. 因此, 更为理想的解决方案是考虑为 BPEL 组合服务引入动态调整的能力, 从而能够在不影响业务功能的情况下, 根据环境的变化调整组合服务与其成员之间的绑定关系.

3　基于运行时体系结构的 BPEL 支撑环境

3.1　运行时体系结构

软件系统的体系结构一般是指该系统的构件组成, 构件间的相互关系和连接方式, 所形成的系统结构配置, 及其动机, 模式和约束等, 它本身原本是一个较为抽象的规约, 最终分散而隐含地体现在系统各个构件及其间的交互行为中[6].

基于软件体系结构实现系统的动态演化被认为是一种较为可行有效的技术途径, 其原因在于体系结构可以从全局观察系统的当前配置状态, 从而成为动态演化的重要依据, 而运行期体系结构约束的(可能)破坏往往是动态演化的驱动因素, 对体系结构进行检查也能够从系统层面约束动态演化, 确保演化前后的一致性和完整性[7].

从体系结构的视角观察 BPEL 服务组合方式可以发现, 把 BPEL 组合服务与其成员联系起来的是"伙伴关系", 而正是定义在此关系上的方法调用最终实现了组合服务的体系结构. 因此为了实现组合服务的动态演化, 必须从体系结构的层面重新考虑上述 partner 的语义. 这里面有两个关键的问题:

(1)组合服务的 partner 虽然定义了其与各个成员之间的交互关系, 但是这种交互关系在运行期却隐含在 BPEL 流程的执行中, 难以控制和修改. 因此需要能够将这些 partner 引用显式地加以集中和结构化的表达, 确保运行期具有对其进行修改的能力.

(2)如何将体系结构层面对 partner 的修改反映到组合服务中? 我们认为, 较为有效的途径是使 partner 的含义不再是一个任意的值, 而看作定义在体系结构上的"函数", 其含义可根据当前的体系结构配置动态确定:

partner= f(architecture);

此时, 若我们对体系结构施加修改, 则会造成对组合服务中伙伴关系的重解释, 从而能够使体系结构的修改反映到组合服务中, 实现动态演化.

基于以上分析, 为支持组合服务的动态演化, 我们为其引入"运行时体系结构". 在这一概念下, 体系结构层面的信息将不再仅仅是一个抽象的规约, 而表现为一个运行期可操控的对象实体. 与一般的对象实体一样, 它具有状态和行为, 特别的, 在这里状态是指体系结构的一些静态特性, 包括构件, 连接子以及它们之间的拓扑关系等. 而行为主要是指系统在运行期所做出的修改行为, 包括构件增加, 删除, 替换等动作. 因此, 组合服务与其成员的伙伴关系可在该对象的状态中得到表达, 而动态演化将具体化为该对象运行期的修改行为. 为使对体系结构层面的修改在组合服务中得以实施, 运行时体系结构对象将成为组合服务与其成员之间交互的中介, 对其修改会造成交互行为的重解释, 如图 1 所示.

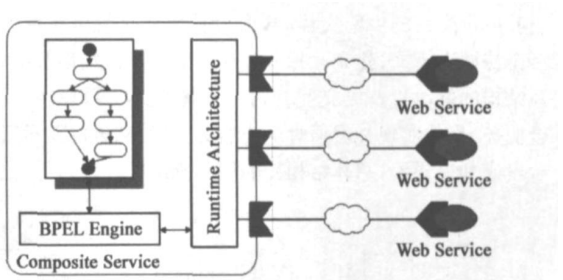


图 1　BPEL 组合服务的运行时体系结构

在以下三小节中, 我们将分别讨论体系结构对象的设计, 实施以及动态演化.

3.2　运行时体系结构的设计

软件体系结构的组成元素与 BPEL 组合服务的构成元素可以产生良好的对应关系. 通过体系结构来描述服务组合的

总体拓扑结构,组合服务及其成员将被绑定到体系结构中相应的构件上,而组合服务对某种接口的请求关系则通过构件上的请求端口表达,并利用连接子为其绑定实现该端口的成员服务.在实现上,采用面向对象方式将体系结构各个组成元素实现为相应的类,而总体配置信息最终封装在一个体系结构类中.

另一方面,体系结构在运行期的行为将被封装为体系结构类中相应的方法.我们引入体系结构风格的概念对构件进行类型化,从而确保对于系统中不同类型的构件提供不同的接口,因为某个构件是否能够调用体系结构修改操作,能够调用哪些修改操作,能与哪些构件交互,以什么方式交互均可能因该构件在体系结构中所处地位而异,通过这一做法,能在一定程度上保证体系结构修改的正确性.例如一个简单的 Master-Slave 风格的体系结构类可定义如下:

```
public class MSArchitecture extends RuntimeArchitecture
    implements I_Slave, I_Master {
    //static structure of runtime architecture
    private Architecture architecture;
    //implementations for methods declared in I_Slave
    public Object invokeOnSlave(Object[] params) throws Exception {...};
    //implementations for methods declared in I_Master
    public Command addSlave(Component slave);
    public Command removeSlave(Component slave);
    ......
    //Constuctor
    public MSArchitecture(Architecture architecture) {...};
}
```

在这里,Slave 的行为是处理 Master 分配的计算任务,而 Master 则可以进行增加 Slave 和减少 Slave 的操作.

3.2 运行时体系结构的实施

如何才能使体系结构类在运行期生成的体系结构对象确实定义了组合服务的体系结构,从自省(Reflection)的角度看就是如何在该对象和组合服务之间建立因果互联关系(Causal Connection).按照我们的理解,体系结构对象将成为组合服务与其成员交互的中间实体,从而为它们之间的交互行为提供一个协同上下文,使交互行为都在该上下文中加以解释.对体系结构的修改将造成交互行为的重解释,从而影响实际的组合服务.具体实现过程通过组合服务开发,部署以及运行三个阶段来加以说明,总体结构图如图 2 所示.

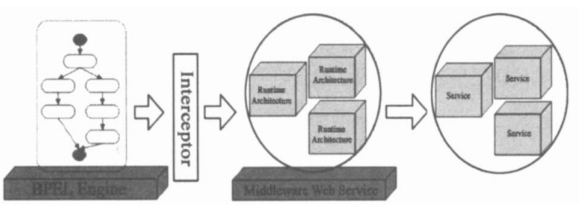


图 2 运行时体系结构实施的总体结构图

开发阶段:不同于通常的 BPEL 组合服务开发,一方面开发者采用 BPEL 刻画组合服务的业务流程.另一方面,开发者通过一个开发工具(通常为可视化的开发工具)进行组合服务体系结构的配置,为便于开发,可以提供一个体系结构风格库,开发者通过直接复用已经定义好的体系结构类或者自定义的体系结构类来配置体系结构,接着从 UDDI 上查找相关的服务并绑定到相应的构件上.

部署阶段:通过将体系结构对象发送到组合服务及其成员所在的主机节点上完成.组合服务并不需要关心成员的物理信息(通常为 URL),而由体系结构对象解释与成员服务的绑定关系.此时体系结构对象将作为组合服务与其成员交互的中间实体,因此需要在组合服务部署前对其输入的成员服务的 WSDL 文件进行修改.为支持这一操作,在所涉及的节点上部署一个特别的 Web Service 作为体系结构的中间件支持,该 Web Service 的结构如图 3 所示.

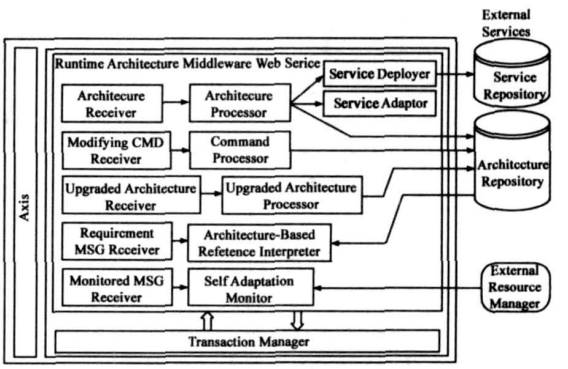


图 3 中间件 Web Service 结构图

该 Web Service 的 URL 将被作为组合服务填入的成员地址信息,从而使对成员的请求转发到该 Web Service 中,在其上匹配合适的体系结构对象并根据体系结构的当前配置解释对成员服务的调用.为了便于体系结构对象的匹配,我们在 URL 之后增加三个字段分别记录与当前的组合服务相匹配的体系结构对象 ID,其对应的构件 ID 和请求方法对应的接口 ID.因此填入成员服务 WSDL 文件中的最终信息如下:

```
〈service name="..."〉
  〈port binding="..." name="..."〉
    〈soap: address location = "URL $ ArchitectureID $ ComponentID $ PortID"/〉
  〈/port〉
〈/service〉
```

运行阶段:该阶段的主要任务是在运行期截获组合服务对其成员的调用,转而由体系结构对象对其进行处理.很显然,为了能够处理各种类型的对成员服务的调用,上述中间件 Web Service 与组合服务交互的接口必须是一种"无类型"的形式,因此我们为其前置一个拦截器,负责收集对成员服务调用的方法名和参数信息,进行相应的封装后转发给中间件 Web Service,然后由其匹配合适的运行时体系结构对象,并参照该对象的当前配置调用与请求接口相绑定的成员服务.

3.4 运行时体系结构的演化

通过将运行时体系结构对象实现为 BPEL 组合服务与其成员交互的中间实体,组合服务的动态演化就可以较为自然

的实施了. 动态演化可视为对体系结构对象的修改行为. 在运行期调用该对象中定义的方法, 可以实现预设的动态演化, 如上述的 Master-Slave 风格的体系结构中, 增加, 去除 Slave 的演化行为被直接定义为 addSlave( ) 和 removeSlave( ), 调用上述方法之后将返回一个(组) 体系结构对象的修改命令, 它们将被发送到各个主机节点上, 主机收到后在满足事务性要求的情况下更新本地的体系结构对象, 从而完成组合服务的动态演化.

另外, 针对设计阶段未预料到的演化动作, 以往的做法是重新开发组合服务的业务流程; 停止旧服务的运行, 再加载新服务, 而本文的运行时体系结构却能对这种类型的演化提供较好的支持, 可以将其理解为体系结构对象的新行为, 并通过定义原有体系结构类的子类实现该新行为, 从而使新行为得以平滑地实施. 比如当一个 Master 负载过重, 需要增加新的 Master 分担负载时, 虽然原有的体系结构风格不支持增加 Master 的操作, 但是可以重新定义以下的体系结构类:

```
public class ExtMSArchitecture extends MSArchitecture
  implements I_ Slave, I_ ExtMaster {
  //New evolution behaviors:
  public Command addMaster(Component master) {...};
  public Command removeMaster(Component master) {...};
  ......
}
```

在运行期将通过一种体系结构"升级" 的方式加载新的体系结构类型, 生成新的体系结构对象, 并用旧体系结构对象的状态来初始化它, 然后利用面向对象的多态机制替换旧体系结构对象, 从而能够调用新定义的方法实现新的演化动作.

4　实例分析

针对于开放环境下组合服务的动态演化问题, 南京大学软件研究所开发了一个动态架构的服务组合平台 ARTEMIS-ARC, 其目标在于为组合服务引入运行时体系结构的支持, 从而能够使用户于开放的环境下动态地组合已有的软件服务, 并进行组合服务的动态演化. 本文提出的 BPEL 支撑环境扩展了该平台的原有实现, 使其支持 BPEL 组合服务. BPEL 执行引擎采用 ActiveBPEL Engine①, 通过将 ARTEMIS-ARC 平台与 ActiveBPEL Engine 引擎相结合, 可以对 BPEL 组合服务的开发, 部署, 运行和演化整个生命周期提供有效的支持.

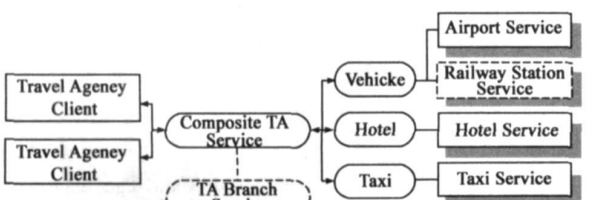


图 4　旅行代理场景示意图

为了展示 BPEL 组合服务动态调整的效果, 我们通过开发一个简单的 BPEL 组合服务实例加以说明. 我们采用了经典的旅行代理的场景. 作为一个旅行代理, 它需要根据用户给出的旅行出发地和目的地向交通运输公司预定交通工具, 并向目的地旅馆预定房间以及向出发地出租车公司预定车辆. 此场景中可能出现的演化行为包括: 出现新的成员服务, 比如出现新的交通运输公司可供用户预定, 我们将该情况设为体系结构的预设性演化. 当旅行代理的业务持续增长, 为均衡负载, 需要开发一个旅行代理的分支服务来接收一部分用户的请求. 我们将该情况设为体系结构的非预设性演化. 该场景的示意图如图 4 所示.

首先, 采用 BPEL 描述该实例的业务流程, 本文的重点在于展示 BPEL 组合服务的动态演化, 因此为简单起见, 我们将该场景理解为一个顺序的流程, 旅行代理依次调用交通运输公司服务, 旅馆服务和出租车服务来完成整个旅行计划的制定. 该过程的大体框架如下:

```
〈process name="travelagency" ...〉
  〈partnerLinks〉
    〈partnerLink name="customer" .../〉
    ...
  〈/partnerLinks〉
  〈variables〉
    〈variable name="travelInfo"
    messageType="tans: planTravelRequest"/〉
    ...
  〈/variables〉
  〈sequence〉
    〈receive name="ReceiveCustomerRequest" .../〉
    ...
     〈invoke inputVariable="queryTicketRequest" name=
"InvokeVehicle"
          operation="queryTicket" outputVariable="queryTi-
cketResponse"
        partnerLink="vehicle" portType="tans: queryTick-
etPT"/〉
    ...
    〈reply name="TravelPlanToCustomer" .../〉
  〈/sequence〉
〈/process〉
```

对于旅行代理组合服务与各个成员服务之间的绑定关系, 通过在 ARTEMIS-ARC 系统的前端集成开发环境中配置体系结构对象来完成, 上述场景可以理解为一个简单的 Master-Slave 风格的体系结构. 在交通工具的配置方面, 首先为旅行代理绑定的是飞机场服务. 如图 5(左) 所示.

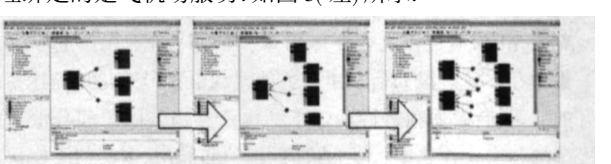


图 5　旅行代理的初始配置 (左), 预设性演化 (中) 和非预设性演化 (右)

① http://www.activepel.org/

配置完成后将体系结构对象发送到组合服务及其成员所在的节点上,当中间件 Web Service 收到体系结构对象后,将在组合服务部署前对其成员的 WSDL 文件进行更改,如上文所述,在其中填入本地中间件 Web Service 的地址以及相应体系结构对象 ID,当前组合服务对应的构件 ID 以及所请求的接口 ID,如下所示:

```
〈service name="..."〉
  〈port name="..." binding="..."〉
    〈soap:address location="http://localhost:8080/
      axis/services/ArchitectureMiddleware$artemis1189380719
$1$S1"/〉
  〈/port〉
〈/service〉
```

当出现新的交通运输公司,比如火车站服务时,我们将其绑定到体系结构相应的构件上,该演化行为相当于增加一个 Slave,从而我们调用 addSlave( ) 完成系统的动态演化. 如图 5 (中) 所示.

当旅行代理的业务持续增长,可以为其开发一个分支服务以均衡负载,由于原有的体系结构类型不支持增加 Master,需对该体系结构类型进行升级,然后将分支服务绑定到新增加的 Master 类型的构件上,从而实现非预设性的动态演化,如图 5(右) 所示.

5 相关工作

组合服务的动态演化问题逐渐成为服务组合技术的一个热点问题. AO4BPEL[8] 采用 AOP(Aspect-Oriented Programming) 扩展 BPEL 以解决其中的 crosscutting concern 问题,实现 BPEL 组合服务的动态演化. WSML[9] 则提出利用 AOP 将 Web Service 管理方面的需求封装为应用程序与 Web Service 之间的一个中间层,以使它们之间的耦合变得松散. 上述两种技术的思路与本文有一定的相似性,然而运行时体系结构为动态演化提供了一个全局的视角,能在更为抽象的层面保证演化行为正确性,同时体系结构类继承的方式也使非预设性演化得以平滑的实施. DySOA[10] 给出了以服务为中心的应用系统在自适应部分的架构准则,与本文的工作有一定的互补性,它的主要关注点不在于如何具体地实施应用系统的动态演化,本文的工作可为解决这一问题提供一个可行的技术思路,而 DySOA 也可以为运行时体系结构中间件中自适应部分的构造提供一定的指导作用.

文献[11] 提出了基于 BPEL4WS 的 Web Services 组合系统的体系结构风格,并针对这种风格设计了体系结构描述语言 WSC/ADL,但与传统软件体系结构的研究类似,该工作仍然集中在设计阶段如何采用合适的规约语言形式化地描述系统的体系结构,而在运行期体系结构信息仍被隐藏和分散在系统的各个构件之中,无法再加以维护和利用. 本文的工作旨在将体系结构的作用延续至系统的运行阶段,通过在运行阶段显式地维护体系结构模型实现系统的动态演化.

在基于软件体系结构实现系统的动态演化方面,CMU 的 Rainbow[12] 是一项具有代表性的工作. 该项目通过使用外置的体系结构管理器实现对体系结构的修改,并借助特定于应用的模型管理机制和属性探测机制以保证体系结构与实际系统的因果互联. 本文的工作采用一种内置运行时体系结构的方式,通过将体系结构转变为运行阶段可操控的对象实体,能够充分利用面向对象语言中的类型、继承、多态等机制,从而为系统动态演化提供一个较为通用的可复用的框架.

6 总结

本文通过分析现有的 BPEL 服务组合方式难以实现组合服务动态演化的原因,提出一种基于运行时体系结构的 BPEL 支撑环境. 通过一个运行时可操控的体系结构对象对 BPEL 组合服务与其成员之间的伙伴关系加以显式地集中和维护,并把体系结构对象实现为组合服务与其成员之间交互的中间实体,使对体系结构的修改造成交互行为的重解释,从而确保组合服务动态演化的顺利实施. 上述的支撑环境在 ARTEMIS-ARC 系统得以实现,并通过在该系统中开发一个组合服务应用实例展示了组合服务动态调整的效果.

参考文献:

[1] Papazoglou M P, Georgakopoulos D. Service-oriented computing: Introduction[J]. Communications of the ACM, 2003, 46(10): 24-28.

[2] Andrews T, Curbera F, Dholakia H, et al. Business Process Execution Language for Web Services, Version 1.1[Z/OL]. http://www-128.ibm.com/developerworks/library/specification/ws-bpel/, 2003.

[3] Curbera F, Khalaf R, Mukhi N, et al. The next step in web services[J]. Communications of the ACM, 2003, 46(10): 29-34.

[4] Yang J. Web service componentization[J]. Communications of the ACM, 2003, 46(10): 35-40.

[5] 马晓星, 余萍, 陶先平, 吕建. 一种面向服务的动态协同架构及其支撑平台[J]. 计算机学报, 2005, 28(4): 467-477.
Ma Xiao-xing, Yu Ping, Tao Xian-ping, Lü Jian. A service-oriented dynamic coordination architecture and its supporting system[J]. Chinese Journal of Computers, 2005, 28(4): 467-477. (in Chinese)

[6] Shaw M, DeLine R, Klein D V, et al. Abstractions for software architecture and tools to support them[J]. IEEE Transaction on Software Engineering, 1995, 21(4): 314-335.

[7] Ma X, Cao J, Chan A, Lü J, Zhang K. A graph-oriented approach to the description and implementation of distributed and dynamic software architecture[A]. The 15th International Conference on Software Engineering and Knowledge Engineering[C]. San Francisco, USA, 2003. 518-524.

[8] Charfi A, Mezini M. Aspect-oriented web service composition with AO4BPEL[A]. Proceedings of the 2nd IEEE European Conference on Web Services[C]. Erfurt, Germany, 2004. 168-182.

[9] Verheecke B, Cibran M A. AOP for dynamic configuration and

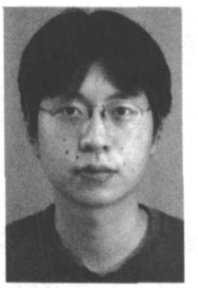
management of web services[A]. Proceedings of the 1st IEEE European Conference on Web Services[C]. Erfurt, Germany, 2003. 137- 151.

[10] Siljee J, Bosloper I, Nijhuis J, Hammer D. DySOA: Making service systems self-adaptive[A]. Proceedings of the 3rd International Conference on Service-Oriented Computing[C]. Amsterdam, Netherlands, 2005. 255- 268.

[11] 杨鑫, 陈俊亮. WSC/ ADL: Web Services 组合系统体系结构描述语言[J]. 软件学报, 2006, 17(5): 1182- 1194.
Yang Xin, Chen Jun-liang. WSC/ ADL: An Architecture Description Language for Web Services Composition System[J]. Journal of Software, 2006, 17(5): 1182- 1194. (in Chinese)

[12] Garlan D, Cheng S, Huang A, Schmerl B R, Steenkiste P. Rainbow: Architecture-based self-adaptation with reusable infrastructure[J]. IEEE Computer, 2004, 37(10): 46- 54.

作者简介:



马　骞　男, 1981 年生, 南京大学计算机科学与技术系硕士研究生, 主要研究领域为 Internet 软件技术、软件体系结构.
E-mail: maqian@ics.nju.edu.cn

虞建杰　女, 1982 年生, 南京大学计算机科学与技术系硕士研究生, 研究方向为 Internet 软件技术、软件 Agent 技术.

马晓星　男, 1975 年生, 南京大学计算机科学与技术系博士, 研究方向为 Internet 软件技术、软件体系结构.
E-mail: xxm@ics.nju.edu.cn

吕　建　男, 1960 年生, 南京大学计算机科学与技术系博士、教授、博士生导师, 研究方向为软件自动化、并行程序形式化方法、面向对象语言和环境.