一种利用业务服务抽象提升服务可用性的方法

温 彦[1),2)] 房 俊[2)] 刘 晨[2)]

[1)](中国科学院研究生院 北京 100190)

[2)](中国科学院计算技术研究所软件集成与服务计算研究分中心 北京 100190)

摘 要 互联网上 Web 服务资源具有自治性和动态性,单个资源的可用性不尽确定.汇聚同类资源、提升抽象层次,并且在运行时动态绑定至具体资源的方式是提升可用性的基本手段之一.文中通过业务服务抽象汇聚相似功能的具体服务,并在此基础上提出了运行时请求拆分和动态切换技术以提升服务的可用性,同时给出了相应的业务服务执行算法.此方法较传统的基于服务副本的方式能够避免单个服务的不可用问题,然而由于服务的可用性状态以及它们在业务能力上的差异,用户请求可能需要通过多个服务的执行才能得以满足,故此方法也带来了服务选择时的额外服务调用开销,因此进而在业务服务执行算法的基础上提出了基于实时可用性更新和服务对用户请求覆盖程度的运行时服务选择算法以提高业务服务的执行效率、减小实现代价.最后使用案例和仿真实验对上述方法进行了评价.

关键词 业务服务;服务;可用性;动态切换;请求拆分

中图法分类号 TP311 **DOI 号**: 10.3724/SP.J.1016.2010.02190

An Approach to Improve the Service Availability on the Basis of Business Service Abstraction

WEN Yan[1),2)] FANG Jun[2)] LIU Chen[2)]

[1)](*Graduate University of Chinese Academy of Sciences, Beijing* 100190)

[2)](*Software Integration and Service Computing Branch Research Center, Institute of Computing Technology, Chinese Academy of Sciences, Beijing* 100190)

Abstract The Web services on the Internet are all autonomous and dynamic in nature, so the availability of a single service cannot be always guaranteed. Clustering resources of the same category and abstracting the resources to a higher level, and dynamically routing to suitable resource at runtime is a common approach to improve availability. In this paper the authors use business service model to cluster services with similar functionalities, and propose an approach combining the request splitting and dynamic switching methods at runtime to improve the service availability. The authors also give the corresponding implementation of the approach. Although by using proposed approach the drawbacks of traditional replication approaches, the possible unavailability status of a single service, have been overcome, extra costs have been introduced because of the services' availability statuses and their different capabilities of covering the business functionality. So the authors propose a service selection algorithm, called SelectIndex, based on services' capability of covering users' requests and an availability calculation method combining real time and historical availability reflection to improve the business service's efficiency and reduce its execution cost. Finally the authors evaluate the above work by case study and simulation experiment.

Keywords business service; service; availability; dynamic switch; request splitting

---

收稿日期:2010-06-08;最终修改稿收到日期:2010-09-01.本课题得到国家自然科学基金(60903137,90812001)、国家"九七三"重点基础研究发展规划项目基金(2007CB310805)、国家"八六三"高技术研究发展计划项目基金(2009AA01Z141)及北京市教育委员会共建项目专项基金资助.**温 彦**,女,1984 年生,博士研究生,研究方向为服务计算、业务服务、服务建模、服务质量. E-mail: wenyan@software.ict.ac.cn. **房 俊**,男,1976 年生,博士,助理研究员,研究方向为服务计算、服务管理. **刘 晨**,男,1980 年生,博士,助理研究员,研究方向为服务计算,业务流程管理.

1 引　言

在以开放、共享和协作为主旋律的互联网计算环境下，Internet 上的资源越来越多地以服务形式对外提供，应用系统的形态也在发生质的变化，不再以固化、独有的形式出现，会包含越来越多的“不为所有，但为所用”的互联网服务构件. 由于互联网服务存在着分散、动态、自治、边界模糊等特征，使得服务使用者面临服务不可控和不确定的问题，从而难以保障基于服务所构造应用的可用性. Kim 等人曾在两年中对发布在 UBR(Universal Business Registry)中的 1000 余个 Web 服务进行了持续监测，结果发现超过 16% 的 Web 服务每周都会失效[1]. 将相同类型资源通过汇聚、提升资源的抽象程度，并在运行时对汇聚资源进行分解并动态路由到具体资源是提升可用性的基本手段之一，本文就是利用已有的业务服务模型作为服务抽象和汇聚手段，提供了一种提升服务可用性的方法.

业务服务是一种 VINCA 服务社区下将动态、开放、无序、无组织的服务集变为可管可控的有序体系的服务组织方式[2]. VINCA 服务社区作为一类特定管理域下构建有序可控服务集合的管理容器，其服务管理的核心思想是设置领域规范界定服务边界[2-3]，通过业务服务方法提供有序化、高可用的服务视图[4]，更重要的是，采用面向可用性的服务全生命周期管理机制，从服务监测[5]、可用性计算以及服务可用性保障等角度确保了互联网服务的可管可控. 业务服务抽象是服务社区的核心元素之一，它基于领域稳定性原理[6]构建，能够反映领域内稳定的业务功能，并实现了对具体服务(本文所述的具体服务包括基于 SOAP/WSDL 协议的 Web 服务、基于 HTTP 协议的 Rest 服务等不同实现手段的服务)的封装，使其具有了易于使用和重用的特点；同时由于业务服务抽象能够聚合功能相似的具体服务，从而使其亦可提供一定的服务可用性提升能力. 本文在服务社区以及业务服务抽象的已有工作基础上，重点讨论一种基于业务服务抽象的提升服务可用性的方法.

可用性通常被定义为系统能够正常工作的概率[7-8]，在不同的上下文和目标中涵义不同，常见的可用性描述方式包括描述在一段时间内系统的整体运行状态的平均可用性，描述系统某一时刻运行状态的瞬时可用性，描述系统在稳定状态下的运行情况的稳态可用性. 本文主要关注用户在服务调用中是否能够成功获得其所需的结果，利用概率的方法来预测某一时刻服务调用是否成功是常用的可用性描述方法，亦能体现业务服务的可用性提升效果，我们在本文中将可用性定义为在某一时刻根据用户请求能够成功返回结果的概率.

当前提升服务可用性的基本手段包括相同服务的冗余方法[9-10]和相似服务的替换方法[11-13]，前者是服务提供者保障服务可用性的主要手段，它从服务提供者直接控制的角度出发，利用容错、备份复制、冗余服务等方法提高可用性，这种方法提供了对单一服务的可用性保障，然而它在开放和自治的互联网环境下所能提供的服务可用性保障能力有限，其主要缺陷在于从整个服务空间来看它依然会有单点失效的风险，当该提供者由于某些原因不再提供服务时，意味着其所有冗余服务均停止提供服务，而基于该服务的组合服务或应用亦会因此失效. 服务替换方法是一种第三方的提升可用性的手段，它基于服务的相似度对比，通过适配方法，用可用服务替换失效服务. 该方法能够避免基于冗余服务方法的单点失效问题，然而当前的服务替换方法在实现服务间自动化转换时常需人工参与适配过程，用户体验较差且在服务替换阶段，应用仍处于失效状态. 利用服务抽象可以有效地克服上述问题. 抽象服务是对功能相似服务的共性特征的抽取和描述，反映了用户的一般性需求；抽象服务聚合了若干功能相似的具体服务，能通过具体服务的执行实现其业务能力，亦能在某一具体服务失效时使用该抽象服务下的其它服务代替之，实现了可用性的提升. 由于抽象服务将服务匹配、异构处理等问题从运行时转移到了建模时，因而能在运行时实现服务的动态切换，同时可将服务切换过程向使用者屏蔽，服务使用者无需修改其自身模块.

当前对服务可替换性的研究主要分为对服务行为一致性的研究[14-15]和对服务接口一致性的研究[11-13,16-18]. 前者关心服务内部状态流转、时序关系等. 文献[14]中作者用 LTS 模拟 Web 服务的行为，在此基础上定义了不同条件下两个 Web 服务之间的相容性概念. 根据不同的相容性定义，作者又提出了满足不同条件的 Web 服务替换性的概念. 文献[15]中基于服务相容性的描述给出了服务可替换的条件. 服务接口一致性的服务替换研究工作主要

包括基于服务适配器的方法[13,18]和基于服务抽象的方法[11-12,16-17].前者的基本思想是通过在被替换的原服务和提供相似功能的可替换服务之间建立映射关系从而生成适配器,这一过程通常通过检验服务间的接口匹配程度来实现的.然而基于在运行时生成服务适配器的方法来进行服务替换很难保证适配的结果完全正确,有时需要人工参与.服务抽象的方法向用户提供一个更通用和灵活的服务接口,同时与若干服务实例绑定,能够将用户请求动态路由到服务实例,由于抽象接口和服务实例间的绑定关系在建模时确定,从而避免了适配器方法在运行时的问题,同时通用接口更利于重用.文献[12]提出了一种基于服务功能多层抽象的可重用组合模式,抽象出具有一般-特殊关系的功能模式层次结构并最终通过服务实例实现,用户可以在所需的层次上进行重用并且可以在已有功能模式的基础上进行逻辑组合构成新的功能模式.然而该方法构建功能模式的逻辑复杂,且功能模式及其层次结构的定义较为随意,重用能力有限.文献[11]提出了一种"服务社区"(不同于我们的社区,该社区是相同功能服务的集合,且具有通用接口定义)的方式组织功能相同的服务,社区提供单一调用接口,基于一些适配规则完成匹配.但是该方法没有给出"社区"及其接口的形成方式,且文中对抽象服务与服务关联的假设约束较强,抽象服务的灵活性较差.文献[17]将文献[11]的社区作为服务可用性保障的手段,利用传统基于副本的可用性保障方法分析了社区方法下的保障策略,指出了传统的被动式和半主动式复制方法不适用于社区方法.文献[16]提出了利用 Infoset 作为通用服务的描述方式,能够映射相同功能的 SOAP 服务和 REST 服务,同时可作为消息处理工具屏蔽底层服务实现技术,该方法旨在解决底层技术的服务抽象问题,对于上层的服务接口异构性未作考虑.

本文所提出的业务服务抽象方法基于领域内稳定的业务活动,更具有一般性和通用性,且充分考虑了不同服务之间的接口异构性,定义了两种服务实例对业务服务的功能覆盖方式,潜在允许绑定更多功能相似的服务,且更具灵活性.上述方法在考虑服务替换时均未将服务可用性作为考量依据,本文在业务服务模型的基础上提出的请求拆分和动态切换方法能够提供双重的可用性提升能力,同时本文提出的基于服务选择因子的业务服务实现算法充分考虑了服务的可用性状态以及服务对用户请求的覆盖程度,且通过仿真实验证明了其具有良好的效果.

2 业务服务提升可用性的原理

2.1 业务服务模型

基于领域稳定性原理构建的业务服务抽象[4]对应了领域内稳定的业务功能,具有易于使用和便于重用的特点.理论上,它可封装多个功能相似的具体服务,这种一对多的关系为提升服务可用性带来了可能,通过在业务服务绑定的多个具体服务中进行运行时切换亦避免了因某个具体服务不可用而造成的应用或组合服务失效的问题.基于上述分析,首先给出业务服务及相关概念的定义.值得注意的是,领域本体体现了领域的稳定性[19],是领域内的共识,基于领域本体构建的业务服务模型更利于理解和重用,因而我们在定义业务服务及其相关概念时普遍使用领域本体概念.

已有工作[4]的业务服务定义包含展示层、特征层、实例层 3 个层面,其中展示层是对业务服务基本功能的描述;特征层详细刻画业务功能的相关业务属性,包括该业务功能对应的输入、输出、服务质量等信息,特征层能够反映业务需求的共性和变化性,同时具有清晰的业务语义;实例层绑定了能够实现该业务服务功能的底层具体服务.本文的业务服务模型为上述定义的简化和改进,包括两层:业务视图层和实例层,前者封装了展示层和特征层,是业务功能向用户呈现的视图,实例层则是业务服务实现的手段.在特征描述上,文献[4]采用了特征树模型,然而由于其过于复杂,不利于用户使用.本文采用特征属性集合的方式来描述业务服务特征,且在可/必选定制方式的基础上引入特征属性的取值域约束以描述具体服务对业务服务的功能定制.

定义 1. 业务服务.将三元组 $businessService=\langle action, entity, features\rangle$ 称为业务服务,它是对领域内的单一业务活动的抽象,其中:$action$ 表示该业务活动的动作,$entity$ 反映该业务活动的客体,即动作的作用对象,二者均来源于领域本体概念,以此来反映其业务语义,业务服务的名称通过 $action+entity$ 的方式进行标识;$features$ 为业务服务的特征属性集合,反映了业务服务的输入、输出、服务质量等各种属性.特别地,定义 $inputs=\{inputf_i \mid i=1,2,\cdots,n\}$ 为业务服务的 n 个输入特征的集合,$outputs=\{outputf_j \mid j=1,2,\cdots,m\}$ 为业务服务的 m 个输出特征的集合,它们均为 $features$ 的子集.

该定义是业务服务的视图层模型,反映业务需

求的共性和个性，支持最终用户使用.

定义 2.　业务服务输入特征. 将四元组 $inputf=\langle name, semantic, sconstraint, range\rangle$ 称为业务服务输入特征，它反映了对业务服务的输入参数、行为特征的描述，其中：*name* 为该输入特征的名称；*semantic* 为该输入所关联的领域本体概念，由此体现业务语义；*sconstraint* 为该特征的可/必选性质，即 $sconstraint \in \{mandatory, optional\}$，表示该特征是否为完成该业务功能必须的输入，*mandatory* 表示该特征为必选输入特征，*optional* 表示该特征为可选输入特征；*range* 为该特征的取值的域约束，表达了该输入特征允许的取值范围.

定义 3.　业务服务输出特征. 将三元组 $outputf=\langle name, semantic, sconstraint\rangle$ 称为业务服务输出特征，它是业务服务执行结果的描述，其中 *name*，*semantic*，*sconstraint* 的涵义与其在输入特征定义中的涵义相同，分别为名称、语义和可/必选性质.

业务服务的必选输入输出特征表达业务功能共性特征，所有与之绑定的服务均实现了对必选特征的支持. 可选特征是用户表达其个性需求的主要方式.

为了获得统一的业务服务抽象视图以及实现用户请求从业务服务向具体服务的转换，业务服务需要屏蔽具体服务在协议、接口、消息等多层面的异构：

在协议层面，存在多种技术协议，例如 SOAP 服务和 Restful 服务；在消息层面，同一语义概念可能表现为不同的模式；在接口层面，不同具体服务所提供的接口数目、顺序、语义也存在区别；

在实现层面，相似的具体服务可能采用不同的实现方法和过程；在实现能力上，相似服务在对业务服务的功能覆盖能力上存在差异，如面向一般问题和特定问题的求解；不同具体服务亦存在服务质量如可用性、可靠性等方面的差距；

而具体服务在安全性、服务契约等层面的异构在社区的有界化管控机制下被自动屏蔽，用户可以在社区的安全保障机制下访问社区内的各类资源.

为了屏蔽相似的具体服务在消息、协议和接口等层面的异构性以及实现具体服务和业务服务的匹配，我们在每个具体服务上加一层封装，称为对具体服务的包装(wrap)，它简化和封装了具体服务的消息和接口，将联系紧密的服务操作进行组合和封装，使不同服务对外呈现一致的接口形式(形式相同，内容不同)，屏蔽与技术相关的细节；同时为包装后的具体服务增加语义等信息. 服务的包装可以借鉴已有的消息解析、转换和接口匹配等方面的工作. 文献[16]提出了一种通用的服务描述方式，能够映射相同功能的 SOAP 服务和 Rest 服务等，同时可作为消息处理工具屏蔽底层服务实现技术，是一种有效的包装服务方法. 我们将经过包装后的具体服务称为虚拟服务，它们具有语义独立的输入输出参数，与业务服务特征实现绑定，同时在业务服务执行时将用户请求还原为服务执行所需的消息和接口格式，通过已有的各类服务执行引擎将请求从虚拟服务转移至具体服务. 具体服务和业务服务的适配过程在服务注册时通过自动或半自动化的方法实现，最终形成具体服务的参数与虚拟服务参数、最终与业务服务特征的映射关系，如图 1 所示.

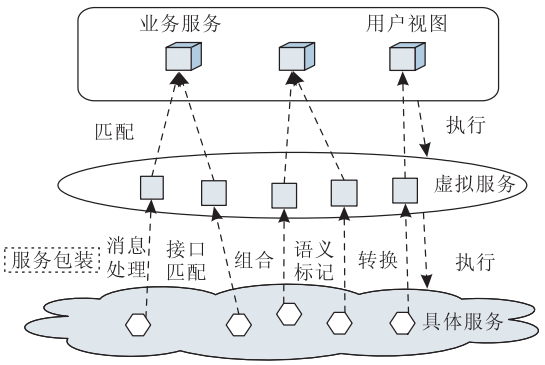

图 1　业务服务的异构处理

本文在虚拟服务之上研究业务服务对服务可用性的提升方法. 虚拟服务的定义如下.

定义 4.　虚拟服务. 将四元组 $service=\langle name, inputs=\{input_i \mid i=1,2,\cdots,n\}, outputs=\{output_j \mid j=1,2,\cdots,m\}, attributes=\{attribute_k \mid k=1,2,\cdots,t\}\rangle$ 称为虚拟服务，它描述了与具体服务相对应的领域内单一的业务功能，其中：*name* 为虚拟服务的名称标识；*inputs* 为虚拟服务 n 个输入参数的集合；*outputs* 为虚拟服务 m 个输出参数的集合；*attributes* 为虚拟服务的 t 个属性的集合，表示该服务是其绑定业务服务的某些特征属性取特定值时的业务功能实现.

定义 5.　虚拟服务输入参数. 将三元组 $input=\langle name, semantic, range\rangle$ 称为虚拟服务输入参数，其中：*name* 为该虚拟服务输入的名称标识；*semantic* 表示该输入所关联的领域本体概念，由此体现业务语义，且与其绑定的业务服务的输入特征的语义概念相匹配；*range* 为该输入所支持的域约束，表达了该输入参数所允许的取值范围.

定义 6.　虚拟服务输出参数. 将二元组 $output=$

〈*name*，*semantic*〉称为虚拟服务输出参数，其中*name*，*semantic*的涵义与其在服务输入定义中的涵义相同，分别为名称和语义.

定义 7. 虚拟服务属性. 将三元组 *attribute*＝〈*name*，*semantic*，*value*〉称为虚拟服务属性，其中：*name* 为该属性的名称标识；*semantic* 表示该属性所关联的本体概念，体现业务语义，且与其绑定的业务服务的输入特征的语义概念相匹配，*value* 表示该属性的取值.

定义 8. 业务服务的实例. 将虚拟服务 *s* 称为业务服务 *bs* 的实例，若 *s* 的参数和属性与 *bs* 的特征属性形成如下的匹配模式：

(1) *s* 的任意输入参数和属性均与 *bs* 的输入特征实现匹配，且匹配两者语义相同.

(2) *s* 的任意输出参数均与 *bs* 的输出特征实现匹配，且匹配两者语义相同.

(3) *bs* 的任意必选输入特征均与 *s* 的输入特征或属性实现匹配，且匹配两者语义相同.

(4) *bs* 的任意必选输出特征均与 *s* 的输出参数实现匹配，且匹配两者语义相同. 图 2 为业务服务及其实例的匹配示例.

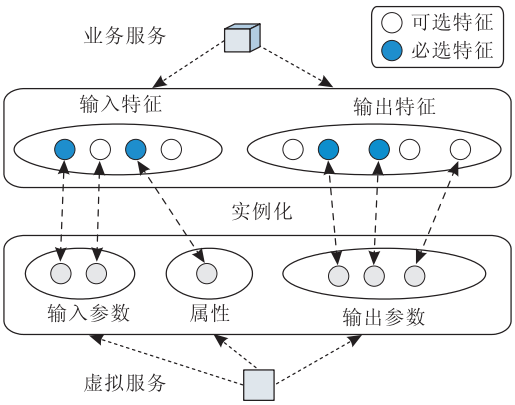

图 2 业务服务与其实例的匹配示例

虚拟服务是屏蔽了消息、接口、协议异构性的具体服务，可以看作是具体服务的功能视图，因此可将两者看作是一个整体，本文若无具体说明，此后将虚拟服务和具体服务统称为服务.

业务服务可在屏蔽了技术异构性的服务间实现动态切换，然而业务服务实现其业务功能的能力依赖于与其绑定的服务对业务服务功能的覆盖能力. 所谓功能覆盖能力是指绑定在业务服务上的服务能够满足用户对业务服务的功能性请求的能力. 举个例子来说明，对于汇率转换类服务，它接受任意两种流通货币类型作为输入参数，输出它们汇率关系，而是否能够实现任意货币的汇率转换，尤其是某些稀有货币种类的汇率转换完全依赖于其绑定的服务是否对其支持，这就构成了服务对业务服务的覆盖能力的差异. 通过调研若干功能相似服务，发现服务对业务服务功能覆盖的常见方式有下面几种.

(1) 服务输入参数和属性与业务服务可选输入特征的绑定以及服务的输出参数与业务服务可选输出特征的绑定反映了服务能够满足用户对业务服务个性化需求的能力.

(a) 例如生物医学领域的“比对序列”业务服务中的可选输入特征——“比对算法”，服务通过其“比对算法”属性来标识自身采用的算法(Blast 或 Fasta 算法)，体现对业务服务该可选输入特征的支持.

(b) 例如天气预报业务服务的一些可选输出特征(日出日落时间、体感温度等)并非被所有服务所支持，一些服务将其输出参数与这些可选特征实现绑定以满足用户对这些特征属性的需求.

(2) 服务输入参数的域约束为业务服务输入特征的域约束的子集.

例如前文所说汇率转换类服务的输入参数中允许的货币种类.

(3) 服务的属性值为业务服务输入特征的域约束的实例.

例如两地距离查询业务服务中的“距离单位”输入特征，服务通过将其属性“距离度量单位”的取值设为“公里”或“英里”来表示其对业务服务此输入特征的覆盖程度.

由上述分析可知，定制可选特征和定制特征的域约束是服务对业务服务功能覆盖的两种主要方式. 而业务服务是否能够通过服务实现其业务能力主要依赖于其绑定的服务是否能够覆盖用户对业务服务的请求.

在此前提下，业务服务的对可用性的提升能力主要表现在两个方面：(1) 用户对业务服务的请求能够同时被多个服务满足，从而可通过动态切换提升可用性；(2) 用户请求不能通过任意一个服务满足时，可通过将用户原始请求拆分为若干子请求分别进行处理，并合并最终结果完成，亦提高了可用性. 我们在 2.2 节讨论这两种方法.

2.2 可用性提升方法

业务服务抽象对服务可用性的提升主要通过如下两种方式实现：

(1) 拆分用户原始请求.

对具体服务进行消息处理后得到的虚拟服务包含语义独立的输入、输出参数，他们之间不存在相互

依赖关系.不同服务的输出参数均包含业务服务的必选输出特征,而对可选输出特征则提供不同程度的功能覆盖.用户对业务服务的复杂请求有时很难通过一个服务的执行实现,而通过将用户的请求拆分为若干子请求分别进行处理,并且将最终结果合并则能够满足用户的初始需求.而通过业务服务抽象,能将上述拆分和合并的过程向用户屏蔽.

拆分的原则为:拆分后的所有子请求均保留原请求的所有输入特征及其取值;原始请求中的所有必选输出特征形成一个子请求的输出特征,不再拆分;可选特征各为一子请求的输出特征.

这样拆分的原因在于所有服务均支持业务服务的必选输出特征,因而不同必选输出特征所对应的服务集合亦相同,而不同服务对于可选输出特征的支持不尽相同,因而将它们各自拆分为一组,以分别计算支持各个输出特征的服务列表.拆分方法如算法 1 所示.

算法 1.　$Split(R)$.

输入:

用户对业务服务 bs 的请求 $R=\langle\{\langle P,V\rangle\},\{O\}\mid P\in bs.inputs, V\in P.range, O\in bs.outputs\rangle$,其中 $\{\langle P,V\rangle\}$ 为输入参数集合,$\{O\}$ 为输出集合.P 为业务服务的输入特征,V 为该特征的取值,O 为输出特征

输出:

子请求集合 $subReqSet=\{R_i\mid R_i=\langle\{\langle P,V\rangle\},\{O_i\}\rangle, O_i\in\{O\}\}$

过程:

```
begin
subReqSet=∅, tempManOutSet=∅;
  for (o in {O})
    if (o.sconstraints==optional)
      subReqSet=subReqSet∪⟨{⟨P,V⟩}, {o}⟩
    else
      tempManSet=tempManOutSet∪{o}
    end if
  end for
subReqSet=subReqSet∪⟨{⟨P,V⟩}, tempManOutSet⟩
end
```

通过请求拆分,请求 R 可通过各个子请求 R_i 的执行并合并最终结果得以实现.

显然可以得到以下结论:能够满足拆分后的各个子请求的服务数量,必然不少于能够满足用户原始请求的服务数量.

(2) 动态切换.

动态切换方法是指,在运行时,将失效服务替换为兼容的可用服务,从而保障用户对服务调用的可用性.判断服务可用状态一般基于请求超时、出错代码等方式.

在业务服务抽象机制下,用户调用业务服务提供的接口,在业务服务绑定的服务中,与用户请求中的输入参数兼容的服务均可作为实现业务服务功能候选服务.

在独立输出拆分的基础上,动态切换的基本单元是拆分后的子请求,能够基于动态切换实现可用性提升主要依赖于能够提供该输出的服务是否能够覆盖用户请求中对输入的约束同时提供请求所需的输出.基于前文的服务对业务服务的功能覆盖能力的分析和定义,则满足用户请求的服务集合的计算方法如算法 2 所示.

算法 2.　$getServiceList(R)$.

输入:

(1) 用户对业务服务 bs 的请求 R 经过请求拆分后得到的某个子请求为:$R=\langle\{\langle P,V\rangle\},\{O\}\rangle$($P$、$V$、$O$ 的涵义与其在算法 2 中相同);(2) 所有与 bs 绑定的服务的集合为 $allServiceSet$

输出:

能够满足 R 的服务集合为 $resultServiceSet$

过程:

```
begin
  resultServiceSet=allServiceSet;
  for (S in allServiceSet)
    for (⟨Pi,Vi⟩ in {⟨P,V⟩})
      //S.Pi表示S中与Pi匹配的输入参数或属性
      if (Vi∉S.Pi.range && Vi≠S.Pi.value
      ||{O}⊈S.outputs)
        resultServiceSet=resultServiceSet−{S}
      end if
    end for
  end for
end
```

由此计算出的服务集合均能满足该子请求,业务服务在此集合上实施动态切换.

在获得了可切换服务集合后,具体的切换方式依赖于 2.1 节所阐述的对具体服务的消息构造和转换、参数匹配和执行方法.此处不冗述.

3　业务服务提升可用性的实现

3.1　DRBE 算法

前文阐述了基于请求拆分和动态切换机制的业务服务提升可用性的原理,然而由于服务的自治性和动态性,业务服务不可预知某一服务在特定时间

的可用性状态,从而采用动态切换方法时可能会由于服务选择策略的不当而带来额外的开销;同时由于采用了请求拆分方法,能够满足子请求的不同服务对用户原始请求的覆盖程度不一,因此服务选择策略不当可能会使业务服务频繁调用对用户原始请求覆盖能力小的服务,从而亦带来额外的服务调用开销.因此业务服务执行过程中服务的选择策略直接影响到业务服务的执行效率和实现代价.影响该策略的主要因素包括:

(1)服务的可用性.当服务不可用时,业务服务通过动态切换机制将请求转移至另一服务执行,若能优先调用高可用性服务,则可最小化对不可用服务的无效调用所带来的额外开销.

(2)服务对用户原始请求的覆盖程度.若能通过一次服务调用获得用户所需的所有输出,则应当避免多次调用带来的额外开销.

对于此二因素,我们采用了综合考虑服务可用性和服务对用户请求覆盖程度的处理方法,3.2 节详述了服务可用性的计算方法;3.3 节基于服务的可用性和服务对用户请求的覆盖程度定义了服务选择因子(SelectIndex)作为业务服务执行时的服务排序和选择依据.

本文将业务服务提升可用性的实现算法称为 DRBE(Dynamic-switching-and-Request-splitting-Business-service-Execution)算法,如算法 3 所示.

算法 3. *DRBE*(*R*).

```
输入:用户对业务服务 bs 的请求 R
输出:请求结果
过程:
begin
//根据算法 2 拆分用户原始请求得到子请求集合
  subReqSet=Split(R)
  for (r in subReqSet) //对于每一个子请求
    if (r.{O} not marked)//若请求 r 的输出未被标记
    //获得可满足该子请求的服务列表
      serviceList=getServiceList(r)
      //对服务列表按服务选择因子排序
      rankOnSelectIndex(serviceList)
      for (s in serviceList)//对列表中的每个服务
        if (s.call()==success)//如果调用成功
          //标记该服务能满足的其它输出
          for (o in s.outputs)
            if(o∈{O})
              markAsCalculated(o)
            end if
          end for
          break;//若 s 调用成功则结束该 r 的执行
        end if
      end for
    end if
  end for
  mergeResultOfOutputs()//将各个输出的结果合并
end
```

3.2 服务可用性的计算

服务的可用性是业务服务的服务执行选择策略的主要依据之一,因此如何计算一个服务的可用性成为利用业务服务实现其可用性保障能力的重要基础.

当前第三方获取计算服务可用性所需的服务运行信息的方式主要基于服务监测.监测可以通过如下几种方式实现:通过截取服务提供者和使用者之间的通信消息实现,该方法实现简单,但会给服务提供者和使用者带来额外负担[20];通过服务提供者提供的监测和管理接口的方式获取服务质量信息,例如服务提供者可以实现 WSDM① 规范所规定的管理接口,这种方式需要服务提供者对服务进行较大修改;通过中介侦测的方式来获取 QoS,例如基于 SOAP 代理、基于中介 log、基于中介事件驱动等方式,这种方法不对现有服务进行修改,结果更客观和可靠[21],但是可能造成对服务性能的影响.本文计算可用性的方法基于监测中介,采用基于已有工作[6]所提出的监测框架.

计算服务可用性时常用的描述稳定系统可用性的监测指标包括平均工作时间 *MTTF*(Mean Time To Failure)和平均失效时间 *MTTR*(Mean Time To Repair).

由此得到系统的失效率 λ 和修复率 μ:

$$\lambda=1/MTTF,\quad \mu=1/MTTR,$$

得到的系统的可用性包括即时可用性(记为 *immediateAvail*)和平均可用性(记为 *averageAvail*)($t=0$表示系统在修复后重新运行的起始时间):

$$immediateAvail=\frac{\mu}{\mu+\lambda}+\frac{\lambda}{\mu+\lambda}e^{-(\mu+\lambda)t},$$

$$averageAvail=\frac{\mu}{\mu+\lambda}.$$

业务服务利用服务可用性及其排序将用户请求分配至可用性较高的服务上,然而业务服务抽象作为一种服务中介和代理,在接受不同用户的大量请

① OASIS. WSDM- Web Services Distributed Management. http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsdm.

求时，基于上述两种可用性计算方式显然不能反映服务当前的运行状态，虽然一般情况下可用性最大的服务成功执行的概率最大，然而亦有可能出现偶然失效情况，而可用性相对较小的服务恰好可用，业务服务应当及时调整请求分配策略，暂时取消在前者上分配请求，而将请求转向后者. 这就是一种实时的可用性更新机制：在 t_0 时刻，业务服务对服务 s 进行了一次调用，且该次调用与上次调用的可用状况相异，若调用成功，则实时可用性（记为 $realtimeAvail$）为

$$realtimeAvail=\begin{cases}1, & t_0<t\leq t_0+MTTF\\ \dfrac{\mu}{u+\lambda}+\dfrac{1}{(t-t_0)\times(u+\lambda)}, & t>t_0+MTTF\end{cases};$$

若调用失败，则

$$realtimeAvail=\begin{cases}0, & t_0<t\leq t_0+MTTR\\ \dfrac{\mu}{u+\lambda}-\dfrac{1}{(t-t_0)\times(u+\lambda)}, & t>t_0+MTTR\end{cases},$$

即当某一时刻对该服务的调用成功时，在 $MTTF$ 内将其实时可用性设为 1，并随着时间的进一步增加，基于 $MTTF$ 逐步降低其实时可用性；若调用不成功，在 $MTTR$ 内将其实时可用性设为 0，并随着时间的进一步增加，基于 $MTTR$ 逐步增加其实时可用性.

3.3　基于服务选择因子的服务选择方法

在 3.2 节可用性计算及排序的基础之上，引入服务对用户原始请求的覆盖程度作为服务选择的策略.

所谓服务对用户原始请求的覆盖程度，是指服务在满足用户请求中输入约束的前提下，能够提供用户所需的输出特征的能力.

定义 9.　设用户请求 r 所需的输出特征个数为 n，服务 s 能够满足该请求对输入特征的要求，且 s 能提供 m 个该请求所需的输出，则 s 对 r 的覆盖程度为 m/n；设服务的可用性为 $s.avail$，则将服务 s 对请求 r 的选择因子定义为

$$SelectIndex=s.avail\times(m/n).$$

需要注意的是，在运行时，已经处理的输出特征的个数不算在 m 内；服务可用性为 3.2 节中所述的服务的实时可用性.

业务服务执行时的服务选择策略依赖于服务的选择因子排序，选择因子越大说明服务的可用性或/和覆盖程度越大，越具有选择价值. 在服务的可用性或覆盖程度为 0 时，基于服务选择因子能够屏蔽不可用和不能提供未处理输出的服务.

4　评　价

4.1　案例分析

我们选择互联网上数量较多的一类服务——天气预报类服务作为本文提出的基于业务服务抽象提升服务可用性方法的分析案例.

案例分析所采用的服务均来源于互联网上的服务存储库或注册中心，例如 programmableWeb[①] 和 WebServiceX[②]、xMethods[③] 等，本文选择了其中 10 个 SOAP 和 Restful（REST）服务来说明天气预报业务服务的可用性提升能力，如表 1 所示. 限于篇幅原因，本文不能穷举全部服务操作，仅使用提供者来区分它们，具体操作可查询服务描述文件或者服务提供者提供的服务说明.

需要注意的是，对于 SOAP Web 服务而言，一个服务包含了若干操作，这些操作相互之间有着密切关联，往往是完成一个业务功能不可或缺的部分，对于 Restful 服务，虽然各个 API 之间可以独立调用，但往往一个服务提供者提供的一组服务亦是紧密相关的. 本文在前文中提到，虚拟服务可以由具体服务组合构成，主要说的是这种情况. 因而本文所指的单一服务是指能够实现业务服务功能的单一服务操作或服务操作组合.

本文主要讨论即时天气查询类服务，我们将其抽象为“即时天气查询”业务服务，简记为 bs. 它包括一个必选输入特征“地点”.

服务 1～10 对该业务服务输入特征的覆盖程度如表 1 所示，对输出特征的覆盖程度如表 2 所示，必选输出特征为“温度”和“天气情况”.

表 2 中亦可看到基于请求拆分的方法获得的关联到各个输出特征的服务列表. 从而用户对业务服务的请求可以拆分为若干子请求，每个子请求的输入参数为原请求的输入参数，每个子请求的输出参数按照输出特征的可必选性质进行拆分，如 2.2 节所述，进而可以分别处理各个子请求，进行结果合并. 例如：用户查询美国某城市的即时天气情况，要求结果能够覆盖所有的输出特征，服务 1～10 均无法直接满足该请求，而通过请求拆分，分别处理所得的各个子请求，再合并各个子请求的处理结果，则能满足该请求.

① www. programmableweb. com
② www. webservicex. net
③ www. xmethods. net

表 1 天气类服务及其对“即时天气查询”业务服务中“地点”特征的覆盖情况

服务序号	服务类型	服务访问地址	操作数	“地点”覆盖情况
1	SOAP/REST	http://www.webservicex.net/globalweather.asmx?wsdl	1	全球
2	SOAP/REST	http://www.webservicex.net/usweather.asmx?wsdl	1	美国
3	SOAP	http://www.deeptraining.com/webservices/weather.asmx?wsdl	1	全球
4	SOAP	http://asyncpostback.com/WeatherService.asmx?wsdl	1	美国
5	SOAP/REST	http://trial.serviceobjects.com/fw/FastWeather.asmx?WSDL	2	美国
6	SOAP/REST	http://api.wxbug.net/weatherservice.asmx?wsdl	3	全球
7	REST	baseurl http://api.wunderground.com	2	全球
8	REST	baseurl http://xoap.weather.com/weather/local	2	全球
9	REST	baseurlhttp://www.worldweatheronline.com/feed/weather.ashx	1	全球
10	REST	baseurl:http://weather.yahooapis.com/forecastrss	1	全球

表 2 不同服务对“即时天气查询”业务服务中基于不同“地点”输入特征的输出特征可选覆盖情况

满足“美国”即时天气查询的服务列表	输出	满足“全球”即时天气查询的服务列表
4,1,2,3,5,6,7,8,9,10	**温度**	1,3,6,7,8,9,10
4,1,2,3,5,6,7,8,9,10	**天气情况**	1,3,6,7,8,9,10
4,1,2,5,6,7,8,9,10	风速	1,6,7,8,9,10
4,1,2,5,6,7,9,10	风向	1,6,7,9,10
4,1,2,5,6,7,9,10	湿度	1,6,7,9,10
4,1,5,6,7,8,10	大气压力	1,7,8,10
4,1,5,7,9,10	能见度	1,7,9,10
4,1,5,6,7	结露点	1,7
4,5,6,7,8	阵风风速	6,7,8
4,2,5,6,10	日出时间	6,10
4,2,5,6,10	日落时间	6,10
4,8,10	气压趋势	8,10
5,7,10	风寒指数	7,10
4,6,8	体感温度	6,8
4,6	月相	6
5,7	热度指数	7
4	紫外线	
5	月出时间、月落时间	
6	最高气压、最低气压、阵风风向、阵风时间、降水概率、室内温度、最高湿度、最低湿度	6

此外,对于各个子请求均存在多个服务能够满足该子请求,从而在此基础上可采用算法 3 的动态切换方法以保障各个子请求的可用性.

以表2为例,我们假设用户对“美国”某一城市的即时天气情况进行查询,并且将表 2 中所有输出特征可能的组合方式进行枚举,每个组合方式代表一个请求的输出,以此来计算通过请求拆分+动态切换、动态切换、单个服务调用 3 种方法能够满足用户请求的服务数,以此来衡量选定方法对可用性的提升能力.在请求拆分+动态切换的方法中,每个子请求可能有若干服务能满足该子请求,取其中最小者作为满足用户原始请求的服务数,即假设 r_1 和 r_2 为 r 的两个子请求,能够满足它们的服务数分别为 10 和 1,则取 1 为能够满足 r 的服务数;对于动态切换方法,取通过单次调用即能满足用户请求的服务的数量;对于单个服务调用方法,若存在能够满足用户请求的服务,则其服务数为 1,否则为 0.将表 2 的最后两行各视为一个输出特征.以此计算出能够满足所有组合方式的服务总数,如表 3 所示.

表 3 能够满足表 2 所有组合方式的服务总数

组合总数	服务数		
	基于请求拆分+动态切换	基于动态切换	基于单个服务调用
131072	152625	15495	12927

表 3 中,基于请求拆分+动态切换的方法能够提供近 10 倍于仅采用动态切换的方法和近 12 倍于单个服务调用的方法所得的能够满足用户所有请求组合的总服务数.虽然不同业务服务的输入输出特征及其定制方式存在差异,从而导致该方法的效果存在差异,但是基于请求拆分+动态切换的方法与仅基于动态切换或单独服务调用的方法相比仍能够显著提高服务的可用性.

4.2 验证实时可用性的计算方法

在业务服务执行时引入服务的可用性作为服务选择的依据,目的在于尽量优先调用可用服务,以避免业务服务对不可用服务的调用以及在他们之间切换带来的额外开销.因此可以使用能够满足用户请求所需调用的服务个数作为评价不同可用性计算方法优劣的依据.

在真实环境下,服务的平均执行时间和平均修复时间常以数百小时为度量单位,在仿真实验中我们将此度量单位缩小,模拟 3 种不同的可用性计算方法的实际效果.方法如下.

在不同的服务器上部署了 5 个相同的 Web 服务实例用来表示它们是绑定在同一业务服务上的能够满足用户请求的服务集合.

创建一个服务控制器随机决定每个服务实例的运行和修复时间,两者均以 3min 为最小单位.

创建单独的监测线程以 min 为单位监测各个

服务实例的运行情况，并且更新基于 3 种方法所得的可用性及其排序情况.

创建客户端以随机方式模拟用户调用，请求频率从数秒到数十秒不等，记录基于 3 种可用性计算和排序的方式能够成功满足用户请求或返回该请求不可被满足结果所需的服务调用次数，以此作为 3 种方法在业务服务执行时服务选择效果的评价标准.

实际测得的结果如图 3 所示，图中展示的是业务服务每处理连续 100 个用户请求所需的服务执行总次数，图中展示了 5 组数据.

图 3 中基于实时可用性的计算方法较基于平均可用性和即时可用性的计算方法服务调用次数平均减少了 32.17%和 39.18%.

图 3 反映了基于实时可用性更新方法较传统的可用性计算方法在业务服务背景下能够通过更少的服务调用成功返回结果，业务服务的实现效率更高.

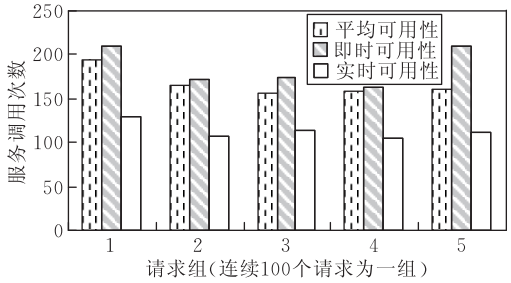

图 3　基于不同的可用性计算方法的业务服务执行时服务调用次数统计

4.3　验证基于服务选择因子的服务选择方法

同 4.2 节一样，我们通过仿真实验来验证服务选择因子方法具有更好的服务选择效果. 服务选择因子一方面能够体现服务可用性的作用，另一方面能够尽可能多地通过较少的服务调用覆盖用户所需的请求，将两者结合的最终目的是为了减少满足用户请求所需的服务调用次数，以减少开销和提高效率. 我们通过计算和比较不采用任何排序手段即随机选择并调用服务、基于服务实时可用性排序、基于服务对请求的覆盖程度排序和基于服务选择因子排序四种方法能够成功满足用户请求所需的服务调用次数来体现服务选择因子的效果. 实验方法如下.

为了说明服务间的异构性，我们在不同的服务器上部署了 10 个不同的 Web 服务实例，它们均绑定在同一业务服务上，该业务服务拥有 10 个可选输出特征，不同的 Web 服务实例支持不同的可选输出特征子集，该子集随机生成，表达了服务对业务服务的功能覆盖程度. 由于服务选择因子的效果主要体现在输出特征上，因此我们假设该业务服务绑定的每个服务均能满足任意请求中对输入的约束，仅在输出特征上体现差异性.

同 4.3 节一样，通过服务控制器控制每个服务实例的随机运行和修复时间；通过监测线程监测并更新服务实例的实时可用性、对请求的覆盖程度和服务选择因子，以及基于上述各个方法的服务排序；客户端随机调用业务服务，不同的是，请求中的输出参数是随机产生的，以反映用户对业务服务的不同需求，记录基于 4 种服务选择方法能够成功满足用户请求或返回该请求不可被满足结果所需的服务调用次数，以此结果作为 4 种方法的服务选择效果的评价标准.

实际测得的结果如图 4 所示，图中展示的是业务服务每处理连续 100 个用户请求所需的服务执行总次数，图中展示了 5 组数据.

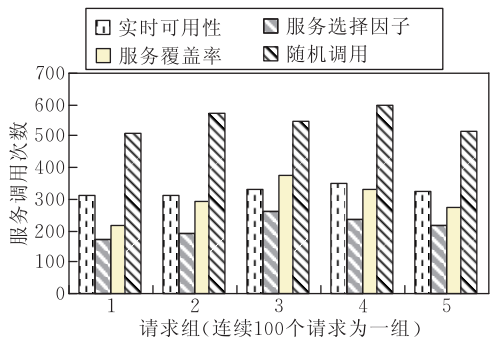

图 4　基于不同的服务选择策略的业务服务执行时服务调用次数统计

图 4 中基于服务选择因子的方法较基于实时可用性的方法、基于服务对用户请求覆盖程度的方法和基于随机调用的方法服务调用次数平均减少了 30.04%、27.49%和 59.62%.

图 4 反映了基于选择因子的方法能够兼顾可用性和服务对用户请求的覆盖程度，具有较好的服务选择效果.

5　结　论

本文提出了一种基于业务服务抽象的服务可用性提升机制. 业务服务模型描述了业务功能的共性和个性特征，能够绑定多个功能相似的异构服务. 业务服务主要通过将用户请求拆分为若干子请求分别

处理并合并结果的方法以及在绑定的服务间实施动态切换的方法实现可用性的双重提升.本文通过案例分析验证了这两种方式能够提供较好的可用性提升能力.在业务服务的实现方法上,本文提出了基于服务实时可用性更新和服务对用户请求覆盖程度的服务选择算法以提高业务服务的执行效率,并通过仿真实验验证了该方法是行之有效的业务服务执行时的服务选择方法.

参 考 文 献

[1] Kim S M, Rosu M C. A survey of public web services. Lecture Notes in Computer Science, 2004, 3182: 96-105

[2] Han Yan-Bo, Wang Gui-Ling, Liu Chen, Zhao Zhuo-Feng. Principles and Practices on Internet Computing-Exploring Essential Problems and Key Technologies Behind the Grid, Cloud and WebX.0. Beijing: Science Press, 2010 (in Chinese)
(韩燕波,王桂玲,刘晨,王菁,赵卓峰. 互联网计算的原理与实践—探索网格、云和 WebX.0 背后的本质问题和关键技术. 北京:科学出版社, 2010)

[3] Wang Zhuo-Hao, Zhao Zhuo-Feng, Fang Jun, Wang Xi-Cheng. A SaaS-friendly service community model and its application in the nationwide service network for sharing science and technology information. Chinese Journal of Computers, 2010, 33(11): 2033-2043(in Chinese)
(王卓昊,赵卓峰,房俊,王希诚. 一种 SaaS 模式下的服务社区模型及其在全国科技信息服务网中的应用. 计算机学报, 2010, 33(11): 2033-2043)

[4] Wang Jian-Wu, Yu Jian, Falcarin P, Han Yan-Bo, Morisio M. An approach to domain-specific reuse in service-oriented environments//Proceedings of the 10th International Conference on Software Reuse (ICSR 2008). Lecture Notes in Computer Science, 2008, 5030: 221-232

[5] Qi Kai-Yuan, Han Yan-Bo, Zhao Zhuo-Feng, Fang Jun. An adaptive service monitor providing runtime extensibility//Proceedings of the 5th IEEE International Symposium on Service-Oriented System Engineering (SOSE). Nanjing, China, 2010: 165-172

[6] Arango G, Prieto-Diaz R. Introduction and overview: Domain analysis concepts and research directions//Domain Analysis and Software Systems Modeling. IEEE Press, 1991

[7] Guo H, Huai J et al. ANGEL: Optimal configuration for high available service composition//Proceedings of the 2007 IEEE International Conference on Web Services. Salt Lake City, UT, USA, 2007: 280-287

[8] Jin J, Nahrstedt K. QoS-aware service management for component-based distributed applications. ACM Transactions on Internet Technology, 2008, 8(3): 1-37

[9] Salas J, Pérez-Sorrosal F, Patiño Martínez M, Jiménez-Peris R. WS-replication: A framework for highly available Web services//Proceedings of the 15th International World Wide Web Conference (WWW'2006). Edinburgh, Scotland, 2006: 357-366

[10] Ye X, Shen Y. A middleware for replicated Web services//Proceedings of the IEEE International Conference on Web Services (ICWS'2005). Orlando, Florida, USA, 2005: 631-638

[11] Taher Y, Benslimane D, Fauvet M-C, Maamar Z. Towards an approach for Web services substitution//Proceedings of the 10th International Database Engineering and Applications Symposium (IDEAS). Delhi, India, 2006: 166-173

[12] Melloul L, Fox A. Reusable functional composition patterns for Web services//Proceedings of the IEEE International Conference on Web Services (ICWS). San Diego, California, USA, 2004: 498-505

[13] Nezhad H R M, Benatallah B, Martens A et al. Semi automated adaptation of service interactions//Proceedings of the International World Wide Web Conference (WWW'07). Banff, Alberta, Canada, 2007: 993-1002

[14] Bordeaux L, Salun G, Berardi D, Mecella M. When are two Web services compatible? TES2004. Lecture Notes in Computer Science. Toronto, Canada, 2005: 15-28

[15] Brogi A, Canal C, Pimentel E, Vallecilo A. Formalizing Web services choreographies. Electronic Notes in Computer Science, 2004, 105(10): 73-94

[16] Li L, Chou W. Infoset for service abstraction and lightweight message processing//Proceedings of the 2009 IEEE International Conference on Web Services. Los Angeles, CA, USA, 2009: 703-710

[17] Maamar Z, Sheng Q Z, Benslimane D. Sustaining Web services high-availability using communities//Proceedings of the 3rd International Conference on Availability, Reliability and Security. Barcelona, Spain, 2008: 834-841

[18] Ponnekanti S R, Fox A. Interoperability among independently evolving Web services//Proceedings of the 5th ACM/IFIP/USENIX International Middleware Conference (MIDDLEWARE). Toronto, Canada, 2004: 331-351

[19] Falbo R de A, Guizzardi G et al. An ontological approach to domain engineering//Proceedings of the International Conference on Software Engineering and Knowledge Engineering. Ischia, Italy, 2002: 351-358

[20] Thio N, Karunasekera S. Automatic measurement of a QoS metric for Web service recommendation//Proceedings of the 2005 Australian Software Engineering Conference. Brisbane, Australia, 2005: 202-211

[21] Sergio Manuel Serra da Cruz, Maria Luiza M Campos. Monitoring E-business Web services usage through a log based architecture//Proceedings of the IEEE International Conference on Web Services (ICWS'04). San Diego, California, USA, 2004: 61-69

WEN Yan, born in 1984, Ph. D. candidate. Her major research interests include service computing, business service, service modeling and quality of service.

FANG Jun, born in 1976, Ph. D., assistant researcher. His major research interests include service computing, service management.

LIU Chen, born in 1980, Ph. D., assistant researcher. His major research interests include service computing, business process management.

Background

Traditional approaches to improve service availability are mainly based on replication mechanisms which are provider-side availability sustaining methods. While these approaches provide high availability for a single service, they are not well applicable for open and autonomous internet environment and will fail under the situation that the provider of the service stop the provision, which will make applications and composed services fail cascade.

When more and more similar services come into emergence, using similar services to replace the failed ones can overcome the above drawbacks, but the challenges shift to how to make services compatible. Business service is a service abstraction model which makes full use of the domain stability and haves taken the diversity of service instances into account to support the common features of domain activities and their individuality. It can be bound with multiple services with similar functionalities which can be used to substitute for failed ones at runtime. This mechanism is call dynamic switch. Meanwhile, with the business service abstraction model, a user's request could be further divided into sub-requests which can be handled separately. The response to the user is a combined result of the responses from all the sub-requests. All the above procedures are hidden from the user, so the user will not feel any interruption.

Another main issue is how to select a most suitable service based on the business service model. Service availability and service's ability to fulfill the user's request are two main factors to achieve high performance of responding, thus an approach taking both factors into account is required.

This work is supported by the National Natural Science Foundation of China under grant Nos. 60903137, 90812001, the National High Technology Research and Development Program (863 Program) of China under grant No. 2009AA01Z141), the National Basic Research Program (973 Program) of China under grant No. 2007CB310805, and the Co-building Special Project of Beijing Municipal Education Commission.