软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2011,22(Suppl.(2)):27−40

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

一种面向语义互操作性的服务聚合方法*

刘建晓+, 何克清, 王 健, 冯在文, 宁 达

(武汉大学 软件工程国家重点实验室,湖北 武汉 430072)

Semantic Interoperability Oriented Method of Service Aggregation

LIU Jian-Xiao+, HE Ke-Qing, WANG Jian, FENG Zai-Wen, NING Da

(State Key Laboratory of Software Engineering, Wuhan University, Wuhan 430072, China)

+ Corresponding author: E-mail: liujianxiao321@163.com

Liu JX, He KQ, Wang J, Feng ZW, Ning D. Semantic interoperability oriented method of service aggregation. ***Journal of Software***, **2011,22(Suppl.(2)):27−40.** http://www.jos.org.cn/1000- 9825/11024.htm

Abstract: In the modern world of service-oriented software engineering (SOSE), the services can be aggregated from the semantic interoperability level to meet the user's personal and diversified needs. First, the paper proposes a service clustering method based on service ontology. It clusters services from the function perspective to form the service clusters. This can significantly reduce the overhead and enhance the service discovery efficiency. In addition, it makes use of the service capability and the interaction information to organize the service clusters form the semantic interoperability level. Furthermore, it discusses the problem of sufficient and necessary capability, and interoperability type. The users can discover the services that can meet their needs efficiently. The corresponding service clustering and discovery algorithms are also designed. Finally, the feasibility and effectiveness of the proposed methods are validated through experiments and a practical case study.

Key words: SOSE; semantic interoperability; service ontology; service clustering; aggregation

摘 要: 在现代面向服务的软件工程(SOSE)时代中,需要面向用户需求从语义互操作层次实现服务的聚合组织来满足用户的个性化多样化需求.首先提出一种通过服务本体引导服务进行聚类的方法,对服务实现功能性聚类形成不同的服务类簇,这样在很大程度上可以降低服务查找空间,提高服务发现效率.在此基础上,从服务执行能力和服务间信息交互两个方面对各服务类簇实现语义互操作层次的组织,探讨了互操作双方满足语义互操作的充分必要能力和互操作类型等问题,使用户可以高效地查找到满足需求的相关联服务.给出了相应的聚类和服务查找算法,最后通过实验和实际案例研究验证所提出方法的可行性和有效性.

关键词: SOSE;语义互操作;服务本体;服务聚类;聚合

近年来服务计算成为业界关注的热点话题,作为服务计算关键性支撑技术之一的 Web 服务是一种运行在 Web 上的自包含、模块化的网络应用程序.为实现以用户为中心,满足涉众服务的高体验质量(QoE)需求,不仅要提供给用户一个或几个离散的服务,还要提供一组具有语义互操作性的服务集作为解决方案,对单个的 Web 服务进行组合[1]来形成组合服务,更好地满足用户需求.

然而随着 Web 服务应用的日益增多,用户有着多样化,个性化的需求,这些需求一方面可以归结为针对同一

* 基金项目: 国家自然科学基金(60970017, 61100017); 国家重点基础研究发展计划(973)(2007CB310801); 中央高校基本科研业务费专项资金(20102110101000118)

收稿时间: 2011-05-02; 定稿时间: 2011-11-07

目标,不同用户会选择不同的服务组合流程.另一方面,由于不同服务的各种质量参数 Qos(如执行时间,费用,可靠性等)值不尽相同,甚至差别较大,这样在面对大量实现相同功能而具有不同 Qos 值的 Web 服务时,有着不同偏好的用户选择结果会有所不同.为解决上述问题,现有一些研究方法将实现相同功能,而具有不同 Qos 参数值的服务进行聚类,然后对聚类的服务进行组织[2−4],形成面向领域共性服务需求的服务资产,为满足用户的个性化需求奠定基础.在此基础上,用户根据其个性化需求在聚合服务中可以直接查找到满足 Qos 期望值的服务.另外,目前互联网环境中的各种服务具有“烟囱林立”,“松散耦合”的分布、异构、自主等特点,需要从语义互操作性的角度来考虑服务聚合组织,使得组织的服务间具有语义互操作性协作能力,这样可以提高服务发现的效率和准确率,从而有效地支撑实现按需服务[5].目前此方面的一些研究方法没有考虑服务的异构、自主等特点,也未从语义互操作的角度对相关的服务进行聚合组织.另外,目前现有的一些服务聚类方法所耗时间比较长,导致效率低下,其他一些方法的准确度则不够高.

针对上述问题,本文的主要工作如下:

(1) 提出一种通过服务本体(见第 2.1 节)引导服务进行聚类的方法,对不同服务者提供的实现相同功能而具有不同 Qos 值的服务进行聚类,形成不同服务类簇.这样在查找服务时,可以快速定位到所需服务,缩小了服务查找空间,提高了服务发现的效率.

(2) 在服务聚类的基础上,从服务执行能力和语义互操作信息交互两个方面对各服务类簇进行聚合组织,给出了相应的服务本体语义互操作性聚类算法,探讨了互操作双方满足语义互操作的必要能力、充分能力和互操作类型等问题.

(3) 利用相关实验对服务聚类方法的时间和准确率等进行验证分析,并通过一个实际案例研究对所提出的语义互操作性聚类和服务查找算法过程进行说明.

本文第 1 节介绍相关概况.第 2 节给出具体服务聚类计算过程.第 3 节详细介绍服务本体语义互操作性聚类算法.第 4 节讨论如何实现服务发现.第 5 节介绍相关工作.第 6 节利用相关实验对所提出方法进行验证分析.第 7 节通过案例说明语义互操作性聚类算法的过程.最后是总结和下一步工作.

1 基本概念与服务聚合框架

1.1 服务聚合

服务聚类:指将不同服务商提供的、实现相同功能、具有相同调用接口的相关服务聚集在一起形成一个服务类簇,同一类簇中的服务具有不同 Qos 值.

服务聚合:指在服务聚类形成不同服务类簇的基础上,将这些服务类簇按照一定的业务逻辑进行集成,构成服务类簇的执行流程.

1.2 语义互操作能力

一般来说,所谓语义互操作能力[6]是指两个软件单元或者系统间交换具有精确含意数据的能力,并且接收方能够准确地翻译或转换数据所携带的信息、信息所携带的知识,即信息、知识能够被理解,最终产生有效的行为协作结果.将语义互操作划分为以下 3 个层次:

含意互操作(meaning interoperability):亦称深度语义互操作,或者完全语义互操作(full semantic interoperability),即双方完全理解协定的语义.

部分语义互操作(partial semantic interoperability):服务实体之间,仅仅能够理解协定的部分语义.

无语义互操作(no semantic interoperability):当服务实体之间的部分语义互操作能力低于一定的阈值时,出现无语义互操作,包括语法(或结构)互操作.

1.3 服务聚合框架

本文提出了一种服务实现语义互操作聚合组织的框架,该框架从以下两个聚类层次实现服务的聚合组织.

(1) 服务聚类:在对同一类实现相同功能的 Web 服务进行建模得到服务本体的基础上,通过服务本体指引

这些具有实现相同功能且不同 Qos 值的服务进行聚类,形成不同服务类簇.这样在实现服务查找时,大大降低服务查找空间,提高效率.

(2) 服务本体语义互操作性聚类:在各种服务本体的基础上,从服务本体间协作需要交换的消息和服务执行对环境产生的影响两个方面抽取出一个互操作的协定本体(见第 3.1 节),该本体是服务本体实施互操作的语义标准.也就是说,对于一个互操作的协定本体,需要多个服务本体之间的行为协作起来才能满足其语义.

2 服务聚类

2.1 相关定义

鉴于各服务存在异质异构性特点,引入本体技术来屏蔽其底层实现技术的差异,从而为实现语义互操作层次的聚合奠定基础.目前本体定义方法缺少对概念状态及状态间转变的描述,文献[7,8]在一般性本体定义方式(O={C,R,rel})的基础上,对其进行扩展生成环境本体[8],该本体可以用表示特定实体状态的类树层次状态机,描述领域实体的状态动态变化特性.本文在上述环境本体定义的基础上,充分利用 hsm 定义的环境实体状态变化对 Web 服务的能力进行表示.

定义 1. 环境实体状态变化:Eff_E={E:prestate→poststate},其中:

- E∈Rsc,可用本体概念表示;
- prestate, poststate∈hsm(E),表示 E 变化前与变化后的两种状态.

例如,信用卡由未支付变为支付状态,表示为 CreditCard: noncharged→charged.车票由 available 变为被预定状态,表示为 Ticket:available→ordered.

定义 2. 状态路径及状态路径长度 Len.

设 e 为环境实体,s_i,s_t∈hsm(e),从状态 s_i 变到 s_t 以及中间需要经过的状态集(S_m)构成了 e 的状态路径,记作 s_i→s_{i+1}→…→s_{t-1}→s_t.e 由状态 s_i 变化到 s_t 的状态路径长度 Len 定义为从 s_i 到 s_t 需要经过的状态个数加 1.

例如,Ticket 的状态路径 available→ordered→sold,则从 available 到 ordered 的状态路径长度为 1,从 available 到 sold 的状态路径长度为 2.当 Ticket 一直处于某状态(如 available)时,则状态路径长度 Len=0.

在上述定义的基础上,本文通过如下定义服务本体来对某一类型的服务进行表示.该本体不指向具体的 Web 服务实例,而只是代表某一类型的抽象服务.

定义3. 服务本体(SO):定义为一个四元组SO={ServiceName,Interface,Capability,Qos}[9].

- ServiceName 表示服务本体的名称.
- Interface={Input,Output},表示包含的输入和输出集,其中:

  Input={IN_i,IN_i∈Rsc,i=0,1,…,inum},

  Output={OUT_o,OUT_o∈Rsc,o=0,1,…,onum}.
- Capability={Precondition,Effect},表示服务执行的前提条件以及执行后对环境产生的影响.其中 Precondition 用环境实体处于某种状态来表示,Effect 用环境实体的状态变化表示:

  Precondition={$Prec_p$, p=0,1,…pnum},$Prec_p$={$entity_p$:$state_p$,$entity_p$∈Rsc,$state_p$∈hsm($entity_p$)},

  Effect={Eff_e,e=0,1,…,enum},Eff_e={$entity_e$:$prestate_e$→$poststate_e$, $entity_e$∈Rsc, $prestate_e$,

  $poststate_e$∈hsm($entity_e$),e=0,1,…,enum}.
- Qos={{$QosName_q$,$Unit_q$,Min_q,Max_q},$QosName_q$,$Unit_q$∈Rsc,q=1,2,3,4},分别表示参数名称,单位,最小最大取值.其中 $QosName_q$ 可以为服务执行对应的时间、代价、可靠性、可用性[10]等.

2.2 服务聚类计算

假设 SO_1 代表特定类型服务本体,WS_2 为进行匹配的具体服务,本文分别从 ServiceName,Interface,Capability 和 Qos 几方面通过加权的方法来计算两者之间的相似度.

(1) 服务名称相似度

目前有多种字符串匹配方法,比如 edit distance,N-gram,Humming distance 等等.本文采用 Edit Distance[11] 方法来计算 Sim(ServiceName$_1$,ServiceName$_2$).

(2) 接口相似度

假设 SO_1 与 WS_2 的 Input 集分别表示为 Input$_1$={IN_{1j},IN_{1j}∈Rsc,j=0,1,…,N_j},Input$_2$={IN$_{2k}$,IN$_{2k}$∈Rsc,k=0,1,…,N_k},其中每个参数可用 Rsc 中概念进行表示.现在已有一些算法计算本体概念间的相似度,本文采用下述算法 match(IN_{1j},IN_{2k})来计算两者间的匹配度[12],见表 1.概念间的推理关系可以直接运用推理机进行计算,例如 Pellet reasoner.表中 Exact,Plugin,Subsume, Intersection,Fail 分别描述了概念间不同的匹配程度,可用定量值来表示上述各匹配值,具体概念间匹配值实例见第 6.2 节.根据 Input1 与 Input2 中包含概念数目的不同,Sim(Input1, Input2)的计算过程见算法 1.

Table 1 The matching between IN$_{1j}$ and IN$_{2k}$

表 1 IN$_{1j}$ 与 IN$_{2k}$ 的匹配度

匹配结果	意义
Exact	IN$_{1j}$≡IN$_{2k}$
Plugin	IN$_{1j}$⊂IN$_{2k}$
Subsume	IN$_{1j}$⊃IN$_{2k}$
Intersection	IN$_{1j}$∩IN$_{2k}$
Fail	IN$_{1j}$ψIN$_{2k}$

算法 1. Input 相似度计算 Sim(Input$_1$,Input$_2$).

输入:Input$_1$={IN$_{1j}$,IN$_{1j}$∈Rsc,j=0,1,…,N_j},Input$_2$={IN$_{2k}$,IN$_{2k}$∈Rsc,k=0,1,…,N_k};

输出:Input$_1$,Input$_2$ 之间相似度.

1. If N_j= =0 Then
2. 返回 1
3. Else If N_k= =0 && N_j≠0 Then
4. 返回 0
5. Else If N_j≤N_k Then
6. 返回 matchpair(Input$_1$,Input$_2$)
7. Else
8. 返回 matchpair(Input$_2$,Input$_1$)

算法 1 中步骤 6 的 matchpair(Input$_1$,Input$_2$)定义为:∀IN$_{1j}$∈Input$_1$,

(¬∃IN$_{2k}$∈Input$_2$⇒IN$_{1j}$∩IN$_{2k}$∨IN$_{1j}$⊃IN$_{2k}$∨IN$_{1j}$⊂IN$_{2k}$∨IN$_{1j}$≡IN$_{2k}$)∨(IN$_{1j}$ΨIN$_{2k}$),则 Val$_{1j}$=0;

(¬∃IN$_{2k}$∈Input$_2$⇒IN$_{1j}$⊃IN$_{2k}$∨IN$_{1j}$⊂IN$_{2k}$∨IN$_{1j}$≡IN$_{2k}$)∨(∃IN$_{2k}$∈Input$_2$⇒IN$_{1j}$∩IN$_{2k}$),则 Val$_{1j}$=0.4;

(¬∃IN$_{2k}$∈Input$_2$⇒IN$_{1j}$⊂IN$_{2k}$∨IN$_{1j}$≡IN$_{2k}$)∨(∃IN$_{2k}$∈Input$_2$⇒IN$_{1j}$⊃IN$_{2k}$),则 Val$_{1j}$=0.6;

(¬∃IN$_{2k}$∈Input$_2$⇒IN$_{1j}$≡IN$_{2k}$)∨(∃IN$_{2k}$∈Input$_2$⇒IN$_{1j}$⊂IN$_{2k}$),则 Val$_{1j}$=0.8;∃IN$_{2k}$∈Input$_2$⇒IN$_{1j}$≡IN$_{2k}$,则 Val$_{1j}$=1;

则 matchpair(Input$_1$,Input$_2$)的值表示为 $1/N_j \times \sum_{j=1}^{j=N_j} Val_{1j}$,该值是求得 Input$_1$ 中的概念在 Input$_2$ 中的最大匹配度,再求和,然后取平均值.根据上述方法可求得 matchpair(Input$_2$,Input$_1$)的值.

与 SO$_1$ 与 WS$_2$ 的 Input 集的相似度计算方法类似,可求得 Sim(Output$_1$,Output$_2$).

(3) 能力相似度

假设 SO$_1$ 与 WS$_2$ 包含的 Precondition 中的 Prec$_p$ 分别表示为 Prec$_{1p}$=entity$_{1p}$:state$_{1p}$,Prec$_{2p}$=entity$_{2p}$:state$_{2p}$,根据文献[7]中对特定实体建立的 THSM 来计算两者之间的相似度 simPre(Prec$_{1p}$,Prec$_{2p}$),见算法 2.

算法 2. Precondition 相似度计算 simPre(Prec$_{1p}$,Prec$_{2p}$).

输入:Prec$_{1p}$=entity$_{1p}$: state$_{1p}$,Prec$_{2p}$=entity$_{2p}$: state$_{2p}$

输出:Prec$_{1p}$, Prec$_{2p}$ 之间相似度

1. If match($entity_{1p}$, $entity_{2p}$)≠Fail Then
2. If $state_{1p}$= =$state_{2p}$ Then
3. 返回 1/2*(match($entity_{1p}$, $entity_{2p}$)+1)
4. Else
5. If 存在由 $state_{1p}$~$state_{2p}$ 的状态路径,且状态路径长度为 Len Then
6. 返回 1/2*(match($entity_{1p}$, $entity_{2p}$)+1/(Len+1))
7. Else
8. 返回 0

算法 2 首先比较 $entity_{1p}$ 与 $entity_{2p}$ 是否相似,若不相似,simPre($Prec_{1p}$,$Prec_{2p}$)的值为 0.否则,当 $state_{1p}$,$state_{2p}$ 是相同状态时,则两者的相似度为 1.当两者不是相同状态时,则根据从状态 $state_{1p}$~$state_{2p}$ 之间的状态路径长度,求得两者间的相似度为 1/(Len+1).然后与实体间的匹配度求和,取平均值返回.例如对于 $Prec_{1p}$=Ticket:ordered,$Prec_{2p}$=Ticket:sold,由于存在 Ticket 的状态路径 available→ordered→sold,则 simPre($Prec_{1p}$,$Prec_{2p}$)=1/2×(1+1/(1+1))=0.75.

Sim($Effect_1$,$Effect_2$)与 Sim($Precondition_1$,$Precondition_2$)的计算方法大致相同,但由于 Precondition 与 Effect 表示方法的不同而有所不同.设 Eff_{1e}=$entity_{1e}$: $prestate_{1e}$→$poststate_{1e}$,Eff_{2e}=$entity_{2e}$: $prestate_{2e}$→$poststate_{2e}$,采用下列 3 个步骤计算 simEff(Eff_{1e},Eff_{2e}).

首先,计算 match($entity_{1e}$,$entity_{2e}$),若该值为 Fail,则返回 0,否则转下一步;

其次,根据 $Prec_p$ 计算方法,计算状态 $prestate_{1e}$ 与 $prestate_{2e}$,以及 $poststate_{1e}$ 与 $poststate_{2e}$ 间的相似度;

最后,取平均值求得 simEff(Eff_{1e},Eff_{2e}).

(4) Qos 相似度

可采用加权的方式计算 Qos_1={$QosName_{1q}$,$Unit_{1q}$,Min_{1q},Max_{1q}}与 Qos_2={$QosName_{2q}$,$Unit_{2q}$,$Value_{2q}$}间的相似度.而 matchqoscluster(Qos_{1q},Qos_{2q})定义为:如果 Qos_{2q}.$Value_{2q}$ 在 Qos_{1q}.Min_{1q} 与 Qos_{1q}.Max_{1q} 范围内时,返回 1,否则返回 0.

通过上述方法,将与同一服务本体进行匹配计算且相似度大于阈值α的相关服务进行聚类,形成实现相同功能且具有不同 QoS 值的服务组成的服务类簇,实现第 1.3 节中所述的第 1 种服务聚类.

3 服务本体语义互操作性聚类

3.1 协定本体

从服务本体实现数据交互的集合和对环境产生的影响,以及两者之间的联系考虑,对协定本体定义如下:

定义 4. 协定本体(CO):定义为三元组 CO={CA,MO,CA_MO},其中:

• CA 表示环境实体的状态变化集合,表示为 CA={Eff_c,c=1,2,…,cnum},Eff_c={$entity_c$:$prestate_c$→$poststate_c$,$entity_c$∈Rsc,$prestate_c$,$poststate_c$∈hsm($entity_c$)}.

• MO 表示服务本体实现语义互操作交换的消息集合,每个消息集中信息可用本体中概念进行表示.表示为 MO={$Mess_i$,i=1,2,…,$m$num},$Mess_i$={con_{ij},con_{ij}∈Rsc,i=1,2,...,$m$num,j=1,2,…,c_i}.

• CA_MO 表示 CA 与 MO 之间的关系集,表示为 CA_MO={$Mess_i$=Eff_j×Eff_k,Eff_j,Eff_k∈CA,$Mess_i$∈MO},用 $Mess_i$ 表示 Eff_j,Eff_j 的实现需要交换的信息.

说明:

(1) 在上述定义中,Eff_c 定义的顺序决定了服务本体间消息交换的顺序,进而决定服务执行的逻辑顺序.

(2) 当 SO_1 与 SO_2 可以通过环境实体的状态变化实现连接但两者间交换的消息为空时,两者间的语义互操作可以通过环境实体状态的变化来体现.

3.2 服务本体语义互操作性聚类

根据协定本体实现服务本体语义互操作性聚类具体过程见算法 3.

算法 3. 服务本体语义互操作性聚类 SOInterCluster.

输入:服务本体集 $SO=\{SO_s,s=1,2,\ldots,snum\}$,协定本体 CO;

输出:实现语义互操作性聚类的服务类簇 Result,服务本体匹配对集 SO_{set}.

1. 初始化 Result=∅,s1,s2(中间变量)=0,SO_{set}=∅,Type(语义互操作类型)
2. Foreach 消息 $Mess_i$ in CO.MO {
3. 根据 $Mess_i$ 在 CO.CA_MO 找到 $Mess_i=Eff_j\times Eff_k$
4. Foreach 服务本体 SO_s in SO
5. Foreach 实体状态变化 Eff_e in SO_s.Effect {
6. If SimEff(Eff_j,SO_s.Effect.Eff_e)≥阈值 Then $s1=s$
7. If SimEff(Eff_k,SO_s.Effect.Eff_e)≥阈值 Then $s2=s$}
8. Result.add(SO_{s1}),Result.add(SO_{s2})
9. If matchMess($Mess_i$,SO_{s1}.Output)≥阈值 && matchMess($Mess_i$,SO_{s2}.Input)≥阈值
10. If SO_1.Output 中 OUT 数目>=SO_2.Input 中 IN 数目 Then
11. If matchCover(SO_1.Output, SO_2.Input, $Mess_i$) Then
12. If SO_2.Input 中 IN 数目≠$Mess_i$ 中 con 数目 Then Type=部分语义互操作 PartialSemanInter
13. Else Type=含意互操作 MeaningInter
14. Else Type=无语义互操作 NoSemanInter
15. Else
16. If matchCover(SO_2.Input,SO_1.Output, $Mess_i$) Then Type=部分语义互操作 PartialSemanInter
17. Else Type=无语义互操作 NoSemanInter
18. SO_{set}.add(⟨SO_{s1},SO_{s2}⟩)
19. Else If matchMess($Mess_i$,SO_{s2}.Output)≥阈值 && matchMess($Mess_i$,SO_{s1}.Input)≥阈值{
20. 类似于步骤 10~步骤 17,判断 SO_{s2} 与 SO_{s1} 实现语义互操作的类型; SO_{set}.add(⟨SO_{s2},SO_{s1}⟩)}}
21. Foreach 服务本体 SO_s in SO-Result
22. Foreach 服务本体 SO_r in Result
23. If matchPrec(SO_s,SO_r) Then {Result.add(SO_s); SO_{set}.add(⟨SO_r,SO_s⟩)}
24. 返回 Result,SO_{set}

上述算法中的 SimEff()是用来计算服务本体与协定本体之间关于环境实体状态变化的语义相似性,计算过程见第 2.2 节.其中 matchMess()的实现与第 2.2 节中的算法 1 相同.步骤 11 中的 matchCover(Output,Input,Mess)用于计算 Output 与 Input 对 Mess 的匹配度.

算法 3 描述了根据协定本体实现服务本体语义互操作性聚类的过程.首先根据协定本体中交换的信息找到对应的实体状态变化,与服务本体的能力进行匹配计算,对匹配的 SO 加入到对应的服务类簇.然后根据 SO 间交换的 Mess,将 SO 中相应服务本体封装成匹配对,实现语义互操作性的分组存储.最后对没有加入服务类簇的服务本体,从服务执行前提与能力方面进行匹配、分组存储.

松散耦合动态协作的服务之间的语义互操作能力刻画[6]包含以下两个方面内容:

(1) 互操作双方是否满足语义互操作的必要能力——语义相似,需要服务本体与协定本体间具有一定的语义相似性.本文是通过计算两者关于环境实体状态的变化来判定是否语义相似的,见算法 3 中的步骤 3~步骤 7.

(2) 互操作双方是否满足语义互操作的充分能力——双方互操作对“协定”语义的理解深度.互操作双方所能够覆盖协定本体的语义深度越大,则双方的语义互操作能力就越强.本文根据 SO 间交换的消息对协定本体语义覆盖度来判断服务本体双方的语义互操作类型,如算法 3 中步骤 9~步骤 20 所示.当 SO_{s1} 的输出可以满足

SO_{s2}的输入且SO_{s2}的输入与所传送的消息部分匹配,或者SO_{s1}的输出部分满足SO_{s2}的输入且SO_{s1}的输出与所传送的消息匹配时,服务本体SO_{s1}与SO_{s2}实现部分语义互操作.当SO_{s1}的输出可以满足SO_{s2}的输入且SO_{s2}的输入与所传送的消息完全匹配时,SO_{s1}与SO_{s2}实现含意互操作.否则两者之间为无语义互操作.图 1(A)中SO_a与So_b实现含意互操作,(B)和(C)中的SO_a与So_b实现部分语义互操作.

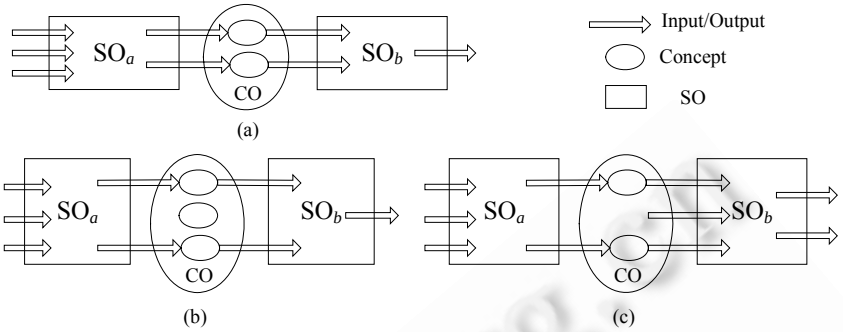


Fig.1 The semantic interoperability type of service ontology

图 1 服务本体语义互操作类型图

4 服务发现

定义 5. 需求(RE):定义为四元组 RE={Re_Capa,Re_Input,Re_Output,Re_Qos}.

Re_Capa 表示用户期望的一些环境实体的状态变化集合,表示为 Re_Capa={Re_Eff_c,c=1, 2,…,re_cnum}, Re_Eff_c={re_entity_c:$re\_prestate_c$→$re\_poststate_c$,re_entity_c∈Rsc,$re\_prestate_c$,$re\_poststate_c$∈hsm(re_entity_c)}.

Re_Input 是为查找服务提供的信息集合,表示为 Re_Input={re_input_i,re_input_i∈Rsc,i=1,2,…,re_inum}.

Re_Output 是用户期望的信息集合,表示为 Re_Output={re_output_o,re_output_o∈Rsc,o=1,2,…,re_onum}.

Re_Qos 为用户期望的服务 QoS 值,表示为 Re_Qos={{$Re\_QosName_q$,Re_Unit_q,Re_Value_q},$Re\_QosName_q$, Re_Unit_q∈Rsc,q=1,2,3,4},每个参数可以用参数名称,单位,取值表示.

上述定义中,Re_Capa,Re_Input 与 Re_Output 主要用于发现满足需求的服务本体,Re_Qos 则主要用于在特定类型的服务类簇内根据 Qos 需求信息查找到合适的服务.算法 4 描述了根据用户特定 RE 请求查找合适服务的过程.

算法 4. 服务查找算法 SerDiscovery.

输入:需求 RE,CO 集合CO_{set},服务类簇 Result,服务本体匹配对集合SO_{set};

输出:WS(满足 RE 的服务集).

1. 初始化 WS=∅,Flag=FALSE,e(实体状态变化序号)=1,SO_m(中间变量)=∅,RIN=RE.Re_Input, ROUT=RE.Re_Output
2. 用户根据需求在CO_{set}中选择合适的协定本体CO_c
3. Foreach 实体状态变化 Re_Eff_e in RE.Re_Capa {
4. Foreach 实体状态变化 Eff_c in CO_c.CA
5. If SimEff(Re_Eff_e, Eff_c)>阈值 Then {
6. 根据 CO_c.CA.Eff_c 在 Result 中找到对应的服务本体 SO_s;SO_m=SO_s; Flag=TRUE
7. Foreach 服务 $service_{si}$ in SO_s 对应的服务类簇
8. matchvalue=Matchqos(RE.Re_Qos, $service_{si}$.Qos)
9. 选择最大的 matchvalue 对应的服务 $service_{si}$,WS.add($service_{si}$)
10. If ROUT⊆$service_{si}$.Output Then 返回 WS
11. Else{

12. RIN=RIN-{service$_{si}$.Input}+{service$_{si}$.Output}; ROUT=ROUT−{service$_{si}$.Output}}}}

13. If Flag Then

14. Foreach 服务本体匹配对 〈SO_{s1},SO_{s2}〉 in SO_{set} {

15. Flag=FALSE

16. If SO_{s1} = = SO_m Then

17. SO_m=SO_{s2}; 在 Result 中找到对应的服务本体 SO_{s2}; Flag=TURE

18. 类似于步骤 7~步骤 12,在 SO_{s2} 对应服务类簇中找到合适的服务加入 WS,并判断是否返回 WS}

19.Else { e++,转步骤 3; Flag=FALSE}

上述算法中 SimEff()的实现过程见第 2.2 节,另外,Matchqos()是用来计算请求服务与所查找服务之间的 Qos 匹配值.

在算法 4 中引入了输入输出集 RIN,ROUT,当请求服务的输出都可得到时,返回找到的服务集 WS.首先选定可满足其需求的协定本体,根据需求中的能力体现(环境实体状态的)找到相应的服务本体,然后在相应的服务类簇中根据 QoS 信息找到满足需求的服务.根据服务本体匹配对,可以快速找到与其具有语义协作关系的服务本体,进而找到相应服务,提高了服务查找的效率.

5 相关工作

目前国内外关于服务聚类已有不少研究,Sudha[13]提出了一种基于服务本体聚类的网格服务发现方法,该方法集中于服务的输入、输出和功能来计算相似性.它主要使用了 agglomerative hierarchical 算法来实现聚类,该方法的复杂性为 $O(n^2)$,其中 n 表示聚类的服务个数.该值明显高于本文所介绍的通过服务本体实现服务聚类方法的复杂性,另外该方法未考虑服务的 QoS 信息.文献[14]提出了一种称为 SWSC 的 Web 服务聚类算法,通过将相似的服务进行分组来改进服务发现效率.该方法没有考虑本体概念间的推理关系,从而影响了服务聚类的准确性.该方法同样采用了 agglomerative hierarchical 算法实现聚类,运算复杂性比较高.文献[15]则通过聚类 wsdl 描述的服务来提高服务发现效率且运用了多种聚类算法,但没有采用本体技术来描述服务,进而没有从语义层次来实现服务聚类.孙萍等人提出了一种服务聚类方法来提高服务发现准确率.该方法通过服务功能和过程执行模型两方面实现服务聚类,采用 Petri 网作为 Web 服务过程描述的形式化工具实现服务过程模型聚类.本文则提出了一种通过服务本体引导服务进行聚类的方法,该方法在基本不影响服务聚类准确度的情况下提高了效率.另外考虑了服务的能力与 QoS 信息等方面的信息,并从服务功能与服务本体语义互操作性两方面来实现服务的聚合组织.

关于服务组织聚合方面也有一些研究,文献[2]中的服务结点就相当于本文介绍的服务本体,该方法从服务的业务逻辑集成角度讨论服务聚合,重点讨论了服务聚合流程的多目标优化的服务动态选择问题.同样文献[17]中的服务池与本文讨论的服务本体比较相似,但此方法重点是根据用户的个性化特定需求来实施服务聚合的.另外,根据此方法用户查找到的服务是原子级服务,并没有对组合服务的发现与服务的能力两方面进行考虑.文献[3]中采用了业务用户编程的方式,面向业务用户组织服务资源以及以业务过程为中心进行业务级服务组合,提出了一种“中间相遇”的方法来实施服务聚合,此方法采用了用户编程方式实施服务聚合.胡春华等人[4]提出了用户为中心的基于业务生成图的服务工作流构造方法,在服务聚合基础上采用生成树的方法进行组织.该方法重点从服务执行过程角度对服务进行组织,并没有从服务间的语义互操作和能力层次考虑对服务实现聚合组织.文献[18]提出了一个支持动态服务组合的 Web 服务三层组织模型,按服务包含的操作、各操作的功能和接口参数信息将它们映射到不同的组织单元.该方法没有考虑使用服务间的语义协作关系来组织服务.Boualem[19]等人提出了服务社区概念,通过服务社区来对服务进行组织,可以应用于动态分布式的 Web 执行环境中.在文献[20]中采用数学模型方式,提出了一个 Web 服务管理框架来实现动态业务过程配置,从服务功能角度,面向任务等方面组织服务来满足用户需求.上述几种方法没有从服务聚类的角度进行考虑,使得这些方法不能应对实现相同功能但具有不同 Qos 值的服务的情况,影响了服务发现效率和准确度.

6 仿真实验

6.1 实验环境

软件环境:Windows XP,MyEclipse 6.0,Pellet reasoner,Mindswap OWL-S API(http://www.mindswap.org/2004/owl-s/api/),xampp(http://www.apachefriends.org/en/xampp.html);

硬件环境:CPU 为双 Intel (R) Core (TM)2 i5 CPU 760@ 2.80GHz,内存为 4G;

数据来源:OWL-TC(http://projects.semwebcentral.org/projects/owls-tc/),这个集合包含了超过 1 000 个服务,覆盖了 7 个应用领域:food,weapon,education,communication,medical,economy 和 travel,其中 ontology 包括了 7 个领域的本体集,本文主要在 education 领域进行实验.

6.2 实验与相关分析

实验 1. 服务聚类时间与准确率比较.

Richi 等人在文献[14]中采用 SimX-Y=(TXY)/(TX+TY+TXY)的形式来计算 X 与 Y 集之间的相似度来实现服务聚类,该方法记作 SWSC.一些常规聚类算法可用于服务聚类中,比如 K-means, Binary-Positive[21], Graph-based[22],Hierarchical[13]等,文献[13]采用了 Hierarchical 方法进行聚类.

本实验主要比较 SWSC,Hierarchical 以及本文 SO 方法实现服务聚类的时间和准确率.服务聚类时间是聚类特定数目的服务所用的时间,可见所用时间越小,聚类的效率就越高.本文用式(1)来计算服务聚类的准确度 $accuracy\_rate$,其中 Num_{total} 为需要聚类的服务总数目,Num_{corr} 表示可以正确聚类的服务数目.

$$accuracy\_rate = \frac{Num_{corr}}{Num_{total}} \tag{1}$$

本实验在 OWL-TC 中 services 中的 education 领域中进行,运用其中 ontology 集中定义的概念间语义关系(见第 2.2 节)进行推理,如 education 领域中涉及的部分本体概念及之间语义关系如图 2 所示.

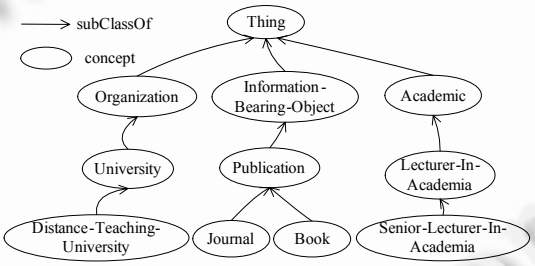


Fig.2 The ontology instance of education domain

图 2 education 领域本体示例

根据表 1,图 2 中概念间的匹配情况见表 2.

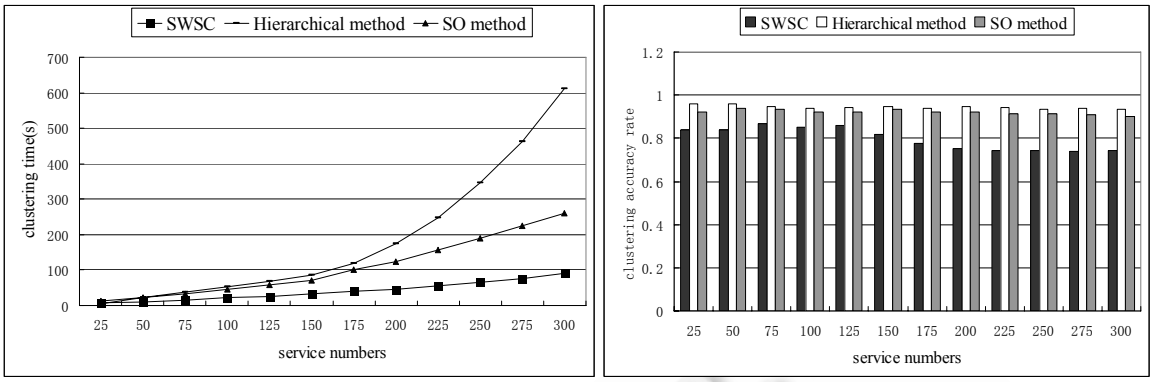
Table 2 The matching of different concepts

表 2 不同概念间的匹配实例

Request	Advertise	Result
Organization	Organization	Exact
Pulication	Book	Plugin
University	Organization	Subsume
Journal	Book	Intersection
University	Academic	Fail

本实验数据是在 OWL-TC 中 services 中的 education(包括 286 个服务)领域集的基础上,手动添加 14 个服务,在共 300 个服务(OWL-S 描述)中进行实验.本实验重在比较服务聚类方法的时间和准确率,同时结合实际情况,OWL-S 形式描述的服务考虑了 ServiceName,Input 和 Output 信息.本实验在服务个数分别为

25,50,75,100,125,150,175,200,225,250,275,300 的情况下进行.设置阈值α(见第 2 节)为 0.6,服务聚类时间与准确率的实验结果如图 3 所示.



(a) 服务聚类时间比较 (b) 服务聚类准确率比较

Fig.3

图 3

通过图 3(a)可得,对于特定的服务聚类方法,聚类时间跟服务数目有关,服务个数越多,聚类时间就越长.但对于特定数目的服务,采用不同的聚类方法,所用时间相差较大.SWSC 聚类方法所用时间最少,而 Hierarchical 方法最多.这是由于 SWSC 方法在进行本体概念间的匹配比较时,只考虑了概念间的相等关系,并没有考虑概念间的语义推理关系,比如 superclassof,subclassof 等,所以 SWSC 的方法所耗的聚类时间最小.而 Hierarchical 方法需要计算每两个服务之间的相似度,使得该方法所用时间最多.在本文的 SO 方法中,相关的服务是通过特定的服务本体来实现聚类的,不需要计算每两个服务之间的相似度,使得该方法所用时间要比 Hierarchical 方法少.另外,SO 方法在对本体概念进行匹配时,考虑了概念间的语义推理关系,使得该方法所用时间大于 SWSC 方法.

在采用上述 3 种方法实现服务聚类的基础上,聚类服务的准确率如图 3(b)所示.可见 SWSC 对应的准确率最低,这是由于该方法仅仅考虑了本体概念间相等关系,并没有考虑概念间的其他语义关系,进而影响概念间的匹配度和服务间相似度的准确性.Hierarchical 方法考虑了概念间的所有语义推理关系并且计算了每两个服务间的相似度,该方法对应的服务聚类准确度最高.由于定义的服务本体会影响相应服务聚类的准确性,本文 SO 方法的服务聚类准确率稍低于 Hierarchical 方法.

综上,SWSC 服务聚类方法的所用时间最少,但准确率也最低.Hierarchical 方法的聚类准确率最高,但它所用时间要远多于其他两种方法.SO 方法在准确率稍低于 Hierarchical 方法情况下,大大缩短了所用时间.

实验 2. 不同参数值下聚类服务个数对比.

本实验在上述实验的 300 个服务基础上,在α(见第 2.2 节)分别取 0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9 情况下,运用 SO 方法对服务进行聚类,对聚类后服务的总个数进行比较分析,实验结果见表 3.

Table 3 The number of clustered services under different parameter values

表 3 不同参数值下聚类服务个数对比

不同α值	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
聚类的服务个数	295	295	295	291	291	284	284	281	279

在α取值不同的情况下,聚类的服务个数有所不同.随着值的增大,聚类的服务个数逐渐变小.这是由于在对不同服务与服务本体进行相似度计算时,α阈值越大,那些相似度大于该阈值的服务个数就越少,从而使的聚类服务的个数变少.

聚类服务的个数主要在 0.4,0.6 和 0.8 这几个点上发生变化.这与在 match()函数中所设置的 Exact,Plugin,Subsume,Intersection,Fail 的值有关.在本实验中,上述各值分别取为 1,0.8,0.6,0.4,0.例如,图 2 中的 education 领域相关概念间的语义关系,测试服务集的输入/输出对应的概念间的语义匹配度为上述各值.如

university_lecturer-in-academia_service.owls 服务对应的输入/输出概念分别为 University 与 Lecturer-In-Academia,university_senior-lecturer-in-academia_service.owls 服务对应的输入/输出概念分别为 University 与 Senior-Lecturer-In-Academia,University 与 University 之间匹配度为 1,Lecturer-In-Academia 与 Senior-Lecturer-In-Academia 之间匹配度为 0.6,则两个服务之间的匹配度为 1/2×(1+0.6)=0.8.

7 应用案例研究

以旅游安排为例,说明服务本体实现语义互操作性聚类思想和过程,定义以下几类服务本体,SO_t 表示可以提供机票预订功能的服务,SO_h 表示可以提供酒店预订功能的服务,SO_c 表示能提供信用卡支付功能的服务,SO_d 表示提供机票投递功能的服务(此案例中不考虑服务的 QoS 信息).

7.1 协定本体表示

根据协定本体的定义,旅游安排实例相关的协定本体包括以下信息:机票预订,根据到达时间预订酒店,且用信用卡进行支付,另外完成机票的投递,该协定本体见表 4.

Table 4 The presentation of CO

表 4 协定本体表示

Parameters	Expression
CA	Eff_1=Ticket:available→sold; Eff_2=Hotelroom:vacancy→paid Eff_3=CreditCard:noncharged→charged; Eff_4=Ticket:sold→delivered
MO	$Mess_1$=ArriveTime; $Mess_2$=TicketPrice; $Mess_3$=HotelPrice
CA_MO	$Mess_1$= Eff_1×Eff_2; $Mess_2$= Eff_1×Eff_3; $Mess_3$=Eff_2×Eff_3

7.2 服务本体语义互操作性聚类过程

根据算法 3,结合以上关于协定本体的表示,各服务本体实现语义互操作性聚类的过程见表 5.

Table 5 The process of service ontology semantic interoperability clustering

表 5 服务本体语义互操作性聚类过程

Steps	Mess					
	$Mess_1$=ArriveTime		$Mess_2$=TicketPrice		$Mess_2$= HotelPrice	
Effect	Eff_1	$s1=t$	Eff_1	$s1=t$	Eff_2	$s1=h$
	Eff_2	$s2=h$	Eff_3	$s2=c$	Eff_3	$s2=c$
Result	SO_t,SO_h		SO_t,SO_c		SO_h,SO_c	
matchMess	ArriveTime≡ArriveTime ArriveTime∩RoomTime		TicketPrice≡TicketPrice TicketPrice⊂ObjectPrice		HotelPrice≡HotelPrice HotelPrice⊂ObjectPrice	
matchCover	RoomTime∩ArriveTime ArriveTime∩RoomTime		ObjectPrice⊃TicketPrice TicketPrice⊂ObjectPrice		ObjectPrice⊃HotelPrice HotelPrice⊂ObjectPrice	
Type	部分语义互操作 PartialSemanInter		部分语义互操作 PartialSemanInter		部分语义互操作 PartialSemanInter	
SO_{set}	⟨SO_t,SO_h⟩		⟨SO_t,SO_c⟩		⟨SO_h,SO_c⟩	

得到 Result={SO_t,SO_h,SO_c},SO_{set}={⟨SO_t,SO_h⟩,⟨SO_t,SO_c⟩,⟨SO_h,SO_c⟩}.然后根据算法 3 对 SO-Result 中服务本体从执行前提与能力方面进行匹配,匹配过程见表 6.将 SO_d 添加到 Result 中,将⟨SO_t,SO_d⟩添加 SO_{set} 中,最终 Result 中服务本体的语义逻辑执行关系如图 4 所示.

Table 6 The process of service ontology clustering in SO-Result

表 6 SO-Result 中服务本体聚类过程

Steps	Values
SO-Result	SO_d
SO	SO_t
matchPrec	Ticket≡Ticket; sold==sold
Result	SO_d
SO_{set}	⟨SO_t, SO_d⟩

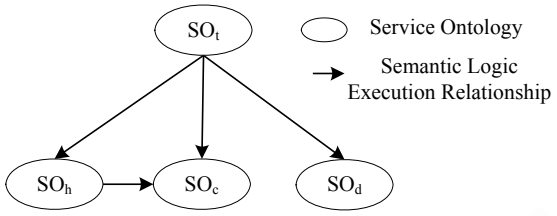


Fig.4 Service ontology semantic logic execution

图 4 服务本体语义逻辑执行图

7.3 服务发现

假设用户提出如下请求:首先要预定机票并需要投递服务,根据飞机到达时间预订酒店,且用信用卡支付机票与酒店预订费用.将上述用户需求表示如下:

Re_Capa:Ticket:available→sold,Ticket:sold→delivered,HotelRoom:vacancy→paid,CreditCard:noncharged→charged;

Re_Input:StartTime,StartLocation,EndLocation,RoomDays,TicketDeliInfor;

Re_Output:PlaneNumber,PayTicketInfor,PayHotelInfor,TicketDeliveredInfor;

通过算法 4,根据用户请求在 Result 中查找满足其需求的服务的具体过程如下:

(1) 根据 Re_Capa 中任一 Re_Eff,如 Ticket:available→sold,通过协定本体查找到 SO_t 具有该能力,根据 Qos 在 SO_t 对应的服务类簇中找到合适的服务;

(2) 更新 RIN=RoomDays,TicketDeliInfor,PlaneNumber,TicketPrice,ArriveTime;

ROUT=PayTicketInfor,PayHotelInfor,TicketDeliveredInfor;

(3) 在 SO_{set} 中查找与 SO_t 具有语义协作关系的 SO:包括 SO_h,SO_c,SO_d;

(4) 对 SO_h,SO_c,SO_d 与 Re_Capa 中其余 Re_Eff 匹配,找到对应的 SO,然后根据 Qos 信息在各 SO 对应的服务类簇中找到合适的服务,返回.

8 结 论

在现代面向服务的软件工程时代中,针对用户需求的多样化、个性化特点,对注册库的服务实现两个层次的聚类:一方面根据建模得到的服务本体引导相关的服务实现功能层次的聚类,形成不同服务本体对应的服务类簇,缩小服务查找空间,提高查找效率.另一方面对服务本体实现语义互操作性聚类,从而对不同服务类簇进行语义互操作层次的组织,实施服务聚合,用户可以查找到满足其需求的最佳服务或者相关联的组合服务,提高了服务查找的效率和准确率.

下一步的研究工作主要包括:根据实验结果反馈自动设定匹配度阈值,从情景等方面对服务的聚合进行考虑,同时进一步从服务本体的 Precondition 与 Effect 方面考虑服务间的语义互操作性.另外还要进行更多的实验,以实际问题来验证本文所提出的方法.

References:

[1] Hwang SY, Lim EP, Lee CH, Chen CH. Dynamic Web service selection for reliable Web service composition. IEEE Trans. on Services Computing, 2008,1(2):104−116. [doi: 10.1109/TSC.2008.2]

[2] Liu SL, Liu YX, Zhang F, Tang GF, Jing N. A dynamic Web services selection algorithm with QoS global optimal in web services composition. Journal of Software, 2007,18(3):646−656 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/18/646.htm [doi: 10.1360/jos180646]

[3] Zhao ZF, Han YB, Yu J, Wang JW. A service virtualization mechanism for business user programming. Chinese Journal of Computer Research and Development, 2004,41(12):2224−2230 (in Chinese with English abstract).

[4] Hu CH, Wu M, Liu GP, Xu DZ. An approach to constructing Web service workflow based on business spanning graph. Journal of Software, 2007,18(8):1870−1882 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/18/1870.htm [doi: 10.1360/jos181870]

[5] Dominique G, Vlad T, Stamatis K, Patrik S, Domnic S. Interacting with the SOA-based internet of things: Discovery, query, selection, and on-demand provisioning of web services. IEEE Trans. on Services Computing, 2010,3(3):2223−2235.

[6] He KQ, Wang J, Liang P. Semantic interoperability aggregation in service requirements refinement. Journal of Computer Science and Technology, 2010,25(6):1103−1117.

[7] Wang PW, Jin Z, Liu L, Cai GJ. Building toward capability specifications of Web services based on an environment ontology. IEEE Trans. on Knowledge and Data Engineering, 2008,20(4):547−561. [doi: 10.1109/TKDE.2007.190719]

[8] Ye RH, Jin Z, Wang PW, Zhen LW, Yang XF. Approach for autonomous Web service aggregation driven by requirement. Journal of Software, 2010,21(6):1181−1195 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/3666.htm [doi: 10.3724/SP.J.1001.2010.03666]

[9] Zeng LZ, Benatallah B, Dumas M, Kalagnanam J, Chang H. QoS-Aware middleware for Web services composition. IEEE Trans. on Software Engineering, 2004,30(5):311−327. [doi: 10.1109/TSE.2004.11]

[10] Zhang PY, Huang B, Sun YM. A Web services matching mechanism based on semantics and QoS-aware aspect. Chinese Journal of Computer Research and Development, 2010,47(5):780−787 (in Chinese with English abstract).

[11] Wombacher A, Rozie M. Evaluation of workflow similarity measures in service discovery. Service Oriented Electronic Commerce, 2006,7(26):51−71.

[12] Li L, Horrocks I. A software framework for matchmaking based on semantic Web technology. In: Gusztav H, Bebo W, eds. Proc. of the 12th Int'l Conf. on World Wide Web. New York: ACM, 2003. 331−339. [doi: 10.1145/775152.775199]

[13] Sudha R, Thamarai SS. Semantic grid service discovery approach using clustering of service ontologies. In: Proc. of the IEEE TENCON. Hong Kong: IEEE Computer Society, 2006. 1−4. [doi: 10.1109/TENCON.2006.343963]

[14] Richi N, Brian L. Web service discovery with additional semantics and clustering. In: Lin TY, ed. Proc. of the IEEE/WIC/ACM Int'l Conf. on Web Intelligence. Washington: IEEE Computer Society, 2007. 555−558. [doi: 10.1109/WI.2007. 112]

[15] Ram S, Hwang Y, Zhao HM. A clustering based approach for facilitating semantic Web service discovery. In: Proc. of the 15th Annual Workshop on Information Technologies & Systems. 2006. 1−6.

[16] Sun P, Jiang CJ. Using service clustering to facilitate process-oriented semantic Web service discovery. Chinese Journal of Computers, 2008,31(8):1340−1353 (in Chinese with English abstract).

[17] Liu XZ, Huang G, Mei H. Consumer-Centric service aggregation: method and its supporting framework. Journal of Software, 2007,18(8):1883−1895 (in Chinese with English abstract). http://www.jos.org.cn/18/1883.htm [doi: 10.1360/jos181883]

[18] Gao Y, Na J, Zhang B, Yang L, Ye L. S-Layer Web services organization model for dynamic service composition. Chinese Mini-Micro Systems, 2006,27(10):1879−1882 (in Chinese with English abstract).

[19] Benatallah B, Dumas M, Sheng QZ, Ngu AHH. Declarative composition and peer-to-peer provisioning of dynamic Web services. In: Proc. of the 18th Int'l Conf. on Data Engineering. Washington: IEEE Computer Society, 2002. 297−308. [doi: 10.1109/ICDE.2002.994738]

[20] Zhang LJ, Li B. Requirements driven dynamic services composition for Web services and grid solutions. Journal of Grid Computing, 2004,2:121−140.

[21] Gelbard R, Goldman O, Spiegler I. Investigating diversity of clustering methods: An empirical comparison. Data & Knowledge Engineering, 2007,63(1):155−166. [doi: 10.1016/j.datak.2007.01.002]

[22] Jain AK, Murty MN, Flynn PJ. Data clustering: A review. ACM Computing Surveys, 1999,31(3):264−323.

附中文参考文献:

[2] 刘书雷,刘云翔,张帆,唐桂芬,景宁.一种服务聚合中 QOS 全局最优服务动态选择算法.软件学报,2007,18(3):646−656. http://www.jos.org.cn/1000-9825/18/646. htm [doi: 10.1360/jos180646]

[3] 赵卓峰,韩燕波,喻坚,王建武.一种支持业务用户编程的服务虚拟化技术-VINCA 聚合服务机制.计算机研究与发展,2004,41(12): 2224−2230.

[4] 胡春华,吴敏,刘国平,徐德智.一种基于业务生成图的 Web 服务工作流构造方法.软件学报,2007,18(8):1870−1882. http://www.jos.org.cn/1000-9825/18/1870.htm [doi: 10.1360/jos181870]

[8] 叶荣华,金芝,王璞巍,郑丽伟,杨夏芬.一种需求驱动的自主 Web 服务聚集方法.软件学报,2010,21(6):1181−1195. http://www.jos.org.cn/1000-9825/3666.htm [doi: 10.3724/SP.J.1001.2010.03666]

[10] 张佩云,黄波,孙亚民.一种基于语义与 QoS 感知的 Web 服务匹配机制.计算机研究与发展,2010,47(5):780−787.

[16] 孙萍,蒋昌俊.利用服务聚类优化面向过程模型的语义 Web 服务发现.计算机学报,2008,31(8):1340−1353.

[17] 刘譞哲,黄罡,梅宏.用户驱动的服务聚合方法及其支撑框架.软件学报,2007,18(8):1883−1895. http://www.jos.org.cn/18/1883.htm [doi: 10.1360/jos181883]

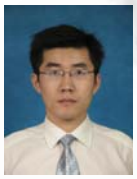
[18] 高岩,那俊,张斌,杨雷,叶蕾.支持动态服务组合的 Web 服务三层组织模型.小型微型计算机系统,2006,27(10):1879−1882.

刘建晓(1984－),男,山西平遥人,博士生,主要研究领域为服务计算,软件工程.

冯在文(1980－),男,博士,主要研究领域为服务计算,需求工程.

何克清(1947－),男,教授,博士生导师,主要研究领域为服务计算,软件工程,需求工程,ISO 国际标准.

宁达(1984－),男,博士生,主要研究领域为需求工程.



王健(1980－),男,博士,主要研究领域为服务计算,需求工程,ISO 标准.