ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.18, No.8, August 2007, pp.1896−1910
DOI: 10.1360/jos181896

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

基于回溯树的Web服务自动组合*

邓水光, 吴 健, 李 莹+, 吴朝晖

(浙江大学 计算机科学与技术学院,浙江 杭州 310027)

Automatic Web Service Composition Based on Backward Tree

DENG Shui-Guang, WU Jian, LI Ying+, WU Zhao-Hui

(College of Computer Science and Technology, Zhejiang University, Hangzhou 310027, China)

+ Corresponding author: Phn: +86-571-87951647, Fax: +86-571-87953079, E-mail: cnliying@zju.edu.cn, http://www.zju.edu.cn

Deng SG, Wu J, Li Y, Wu ZH. Automatic Web service composition based on backward tree. *Journal of Software*, 2007,18(8):1896−1910. http://www.jos.org.cn/1000-9825/18/1896.htm

Abstract: An approach based on backward tree to compose services automatically is proposed. It composes services for a user through three steps: 1) Builds a complete backward trees on-line; 2) Searches for optimal valid generation sources (generation-paths); 3) Composes generation paths. Compared to traditional graph-based methods, it has a smaller search space and avoids the repetition search. Experimental results show that this method has a good performance even the repository has a large number of services.

Key words: SOA (service-oriented architecture); Web service; service composition; backward tree; flow service

摘 要: 在服务规则库的基础上,介绍了回溯树与完备回溯树的概念,并证明了其重要性质.提出了基于回溯树的Web服务自动组合方法.该方法采用分步分治的思想进行服务的自动组合:1) 针对用户请求的输出对象生成完备回溯树;2) 在完备回溯树中选取最佳生成源(生成路径);3) 将生成路径合成为可执行的流程服务.与已有的基于图搜索的自动Web服务组合方法相比,该方法极大地减小了搜索空间,避免了循环搜索,能够满足单目标和多目标的用户请求.仿真实验结果表明,该方法能够在大规模的服务规则库中进行快速的服务组合,从而满足用户请求.

关键词: 面向服务的体系架构;Web服务;服务组合;回溯树;流程服务

中图法分类号: TP311 **文献标识码**: A

Web服务作为互联网中的一种新的计算资源,在电子商务、企业应用集成等领域扮演着越来越重要的角色.近年来,随着Web服务相关标准的持续完善和支持Web服务的软件平台的不断成熟,越来越多的企业将其业务功能和流程包装成标准的Web服务发布出去,实现快速、便捷的寻求合作伙伴、挖掘潜在客户和达到业务增值的目的.然而,如何有效地组合分布于Internet中的各类服务,实现服务之间的无缝集成,形成功能丰富的企业级服务流程以达到企业的商业目标,已经成为Web服务应用的一个关键问题[1].特别是随着面向服务的计算

* Supported by the National Natural Science Foundation of China under Grant Nos.60603025, 60503018 (国家自然科学基金); the National High-Tech Research and Development Plan of China under Grant No.2006AA01Z171 (国家高技术研究发展计划(863)); the National Key Technology R&D Program of China under Grant No.2006BAH02A01 (国家科技支撑计划); the Natural Science Foundation of Zhejiang Province of China under Grant No.Y105463 (浙江省自然科学基金)

Received 2007-02-27; Accepted 2007-05-31

(service-oriented computing,简称SOC)和面向服务的体系架构(service-oriented architecture,简称SOA)在工业界的进一步推广与应用,迫切需要一种有效的服务组合方法来以开发基于SOA的软件系统[2].

近年来,国内外围绕服务组合展开了广泛的研究,提出了众多的组合方法.服务组合相关综述也见诸于国内外学术论文,如文献[3,4].这些方法大多可以分成 3 大类:基于工作流的服务组合方法,如文献[5–7];基于人工智能中逻辑推理、定理证明和任务规划等理论的自动服务组合方法,如文献[8–11];基于图搜索的自动服务组合方法,如文献[12–16].基于工作流的服务组合方法因涉及大量的人工参与,如流程建模、服务绑定、参数设定等,自动化程度不高,服务组合的效率较低;基于人工智能理论的自动服务组合方法虽然摆脱了手工作业,能够实现服务的全自动合成,但这些方法都需要对服务进行预处理和形式化转化,用户使用起来不易掌握,方法的复杂度较高,不易实现;基于图搜索的服务自动组合方法将服务库中服务之间的关系用有向图表达,在图中进行遍历,寻找从输入到输出或者从输出到输入的可达路径,但由于服务库中的服务数量多、服务之间关系纵横交错,使得服务关系图的构建时间开销很大,关系图十分庞大而不易处理.目前,已有的基于图搜索的服务组合方法有的将整个服务库构成的图作为搜索空间,而且在搜索中没有较好的机制来避免循环搜索,因此,目前这些基于图搜索的服务组合方法在服务数量很大、服务间关系复杂的情况下效果不佳.比如,文献[12]从用户提供的输入对象出发,在服务本体关系图中进行遍历,寻找到达用户期望的输出的路径,但该方法仅考虑了从单个输入到单个输出的情况,此外,该方法没有说明如何构造服务本体关系图.事实上,构造服务的关系图是一个非常复杂的过程,特别是在服务数量大且关系复杂的情况下,构造过程将从根本上决定该方法的可用性.文献[15]提出了一种无回溯的反向链算法进行服务合成,与文献[12]相比,该方法无须事先构建整个服务库的关系图,因而其搜索空间相对要小.但该方法在建立输入闭包以及为输出进行回溯时复杂度较高,且方法最后仅得到了用于组合的一个服务集合,而没有进一步说明这些服务之间的调用关系.

本文提出了一种基于回溯树的服务自动组合方法.该方法属于基于图搜索的方法范畴,但不同与已有的基于图搜索的方法.它采用分步分治的思想,通过为用户请求的每一个输出对象即时建立完备回溯树,在建立的过程中选取有效生成路径,最后将路径合成为可运行的组合服务.该方法将搜索的空间受限于回溯树中,大幅度降低了搜索的范围和搜索的复杂度,加快了服务组合的效率.该方法能够处理单目标和多目标的用户请求.仿真实验表明,基于回溯树的服务自动组合方法能够在大规模服务库中快速组合服务,从而满足用户请求.

本文第 1 节介绍服务中的规则与服务规则库的概念.第 2 节提出回溯树与完备回溯树的定义和性质.第 3 节介绍基于回溯树的服务自动组合方法的 3 个步骤.第 4 节是仿真实验与结果分析.最后是总结与展望.

1 服务与规则

Web服务是一组操作(operation)的集合,可以抽象为一个二元组$s=(n,P)$,其中,n为服务的名称,P为该服务的操作集合.操作是Web服务的基本功能实体,Web服务组合最终体现在服务间操作的组合.操作可以表示为一个二元组$p=(I,O)$,其中,I表示该操作接收的输入对象集合,O为操作产生的输出对象集合.因而,操作可以被抽象为一系列输出对象的产生式规则.

定义 1(产生式规则). 给定一个操作$p=(I,O)$和一个输出$o\in O$,则输出o在操作p中的产生式规则形如$I\xrightarrow{p}o$,表示操作 p 将输入对象集合 I 转化成输出对象 o.输出 o 的产生式规则可形式化为一个三元组$r=(O_s,O_d,p)$,其中,$O_s=I$为该规则的源对象集合,$O_d=\{o\}$为该规则的目标对象集合,p为规则对应的操作.

若一个操作能产生 n 个输出对象,则该操作包含 n 条不同的产生式规则,记操作 p 的产生式规则的集合为$\varphi(p)$.将服务内部每一个操作转化成若干产生式规则后,服务$s=(n,P)$可视为规则的集合,而服务库可视为规则库.

定义 2(服务规则库). 服务库$S=\{s_1,s_2,\ldots,s_n\}$对应的服务规则库为$R_S=(I,O,R)$,其中:$I=\bigcup_{s\in S}\left(\bigcup_{p\in s.P}p.I\right)$,为规则库能接收的所有输入对象($s.p$表示服务$s$中的操作$p$,$p.I$表示操作$p$的输入对象集合);$O=\bigcup_{s\in S}\left(\bigcup_{p\in s.P}p.O\right)$,为规则库能

产生的所有输出对象($p.O$表示操作p的输出对象集合),$I\cup O$统称为规则库的对象集合;$R=\bigcup_{s\in S}\left(\bigcup_{p\in s.P}\varphi(p)\right)$,为规则库中所有的规则.

图 1 为服务规则库示意图,左侧为规则库的输入对象集合,右侧为输出对象集合,内部为从输入到输出的转化规则.值得注意的是,在该图中,同样一个对象可以多次出现在右侧,这是因为不同的输入对象组合可以通过不同的规则而产生同一个输出对象.目前,Web 服务还是基于 UDDI 或者类似 UDDI 的服务库进行存储,为了实现基于回溯树的服务自动组合,必须实现服务库到服务规则库的自动转化.给定一个服务库 S,算法 RuleTransformer 将该服务库自动转化为对应的规则库.

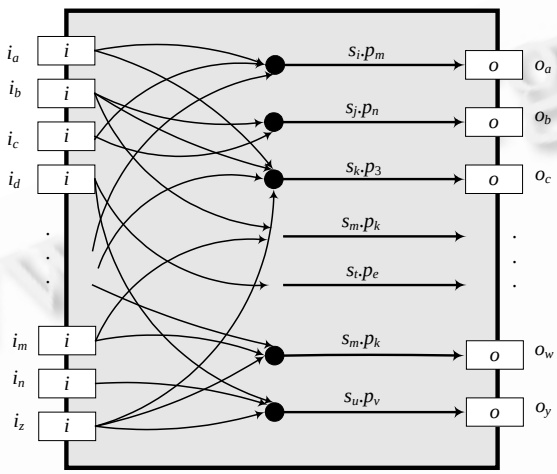

Fig.1 Service rule repository

图 1 服务规则库

算法 RuleTransformer 依次遍历服务库中的每一个服务,针对其操作的输出分别生成一条产生式规则.若服务库中的服务数量为 n,服务包含的最多操作个数为 m,而操作中的最多输出个数为 q,由于 m 和 q 近似为常量,则算法 RuleTransformer 的时间复杂度与空间复杂度均为 $O(n)$.

算法 1. RuleTransformer.

Input: A service repository $S=\{s_1,s_2,\ldots,s_n\}$.

Output: The service rule repository $R_S=(I,O,R)$ generated according to S.

1. **For** each service $s=(n,P)$ in the service repository {
2. **For** each operation $p=\{I,O\}$ in s {
3. $R_S.I=R_S.I\cup p.I$;
4. $R_S.O=R_S.O\cup p.O$;
5. **For** each output $o_i\in p.O$ $(1\le i\le |p.O|)$ {
6. **Create** an generation rule $r_i=(O_{si},O_{di},p_i)$ for o_i where:
7. $O_{si}=I$, $O_{di}=\{o_i\}$ and $p_i=p$;
8. $R_S.R=R_S.R\cup\{r_i\}$;
9. }
10. }
11. }
12. **Return** R_S;

2 回溯树与完备回溯树

2.1 回溯树及其性质

定义 3(回溯树). 给定一个服务规则库$R_S=(I,O,R)$和一个输出对象$o\in O$,该输出的回溯树t_o为一棵以o为根节点的树,可表示为一个三元组$t_o=(V,\chi,E)$,各元素满足如下条件:

(1) $V=\{r\}\cup N$,为回溯树的节点集合,其中:r为根节点,对应输出对象o,用对象集合$\{o\}$作为根节点r的标识;$N=\{n_1,n_2,\ldots,n_m\}$为树中除根之外的节点集合.该集合的每一个元素n由规则库对象集合的一个子集来标识,该子集由函数χ确定,即$\chi(n)\in 2^{I\cup O}$.对于根节点r,$\chi(r)=\{o\}$.

(2) $\chi:V\to 2^{I\cup O}$,为回溯树中节点标识函数,其定义域为回溯树的节点集合V,值域为规则库对象集合的幂集.

(3) $E\subseteq V\times V\times R$为树中的有向边组成的集合,每条有向边均是从树的低层节点指向高层节点.每条边$e\in E$为一个三元组$e=(v_s,v_e,r)$,其中,$v_s\in V$为有向边的起点,$v_e\in V$为有向边的终点,$r\in R$为该有向边对应的规则.

(4) 回溯树的任意一条有向边$e=(v_s,v_e,r)$满足条件:$r.O_s\subseteq\chi(v_s)\wedge r.O_d\subseteq\chi(v_e)$.

(5) 给定回溯树的任意一条以根节点为终点的有向路径$\mathrm{l}=\langle e_i,\ldots,e_j,\ldots,e_k\rangle$(其中,$e_i,e_j,e_k$等为路径 l 覆盖的边,$e_i$为起始边,$e_k$为指向终点即根节点$r$的边),则对于路径 l 上的任意一条边$e$,满足关系:

$$e.r.O_s\cap\left(\bigcup_{e_i\in \mathrm{l}'}e_i.r.O_d\right)=\varnothing,$$

其中,l'为路径 l 上以 e 为起始边的子路径.

条件(1)和条件(2)表明,回溯树的每一个节点均对应一个对象集合,其中,根节点对应输出对象o组成的集合$\{o\}$.条件(3)和条件(4)说明,回溯树的每一条有向边均关联一条规则,该规则将边的起点的对象集合转化成终点的对象集合,因此,回溯树中的任意一个非根节点n与根节点间r的有向路径,均代表了一条可以从n的对象集合转化成r的输出对象.条件(5)避免了回溯树的任意一条路径上出现对象之间的循环转化,即路径上前后出现这样的两条边e_i和e_j,边e_i将对象A转化成B,而边e_j将对象B转化成对象A;此外,条件(5)还禁止出现类似于将对象A转化成aA这样的规则,以此来避免对象的无限转化.为简便起见,此后,$e_i.r.O_s$和$e_j.r.O_d$均省去中间的r,分别表示为$e_i.O_s$和$e_j.O_d$.

以如图 2 所示的服务规则库为例,考察图 3 中的树(如图 3(a)~图 3(c)所示).根据定义可知,图 3(a)是回溯树,而图 3(b)并非回溯树,这是因为在图 3(b)中,由边e_4,e_2和e_1组成的路径存在如下关系:

$$e_4.O_s=\{a,B,f\};\ \bigcup_{e\in \mathrm{l}'=\langle e_4,e_2,e_1\rangle}eO_d=\{E,B,A\};\ e_4.O_s\cap\left(\bigcup_{i\in \mathrm{l}'}e_i.O_d\right)=\{B\}\neq\varnothing.$$

因此,条件(5)不满足.事实上,边e_2根据规则r_2将对象集合$\{D,E\}$转化成$\{B\}$,而边e_4根据规则r_4又将对象集合$\{a,B,f\}$转化成$\{E\}$,因此出现了对象B和E之间的循环转化.同理可以说明图 3(c)并非回溯树.

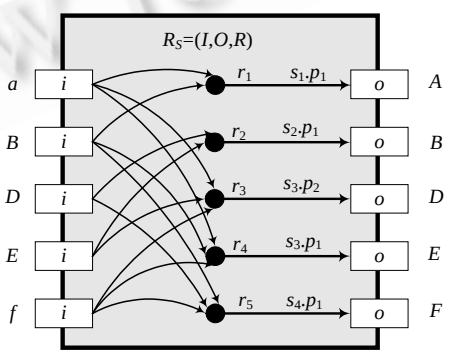

Fig.2 An example of service rule repository

图 2 一个服务规则库示例

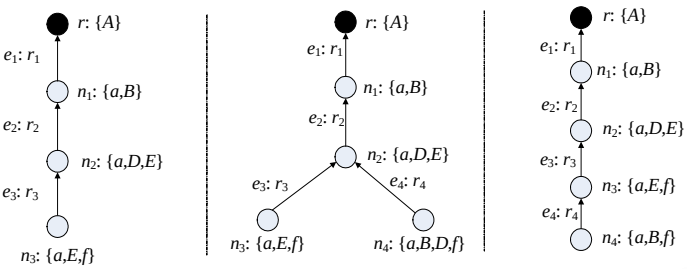

Fig.3 Backward-Tree and non-backward-tree

图 3 回溯树与非回溯树

定义 4(生成路径). 给定输出对象o的一棵回溯树$t_o=(V,\chi,E)$和t_o的一个非根节点v,从v到r的有向路径为输出对象o的一条生成路径,记为有序队列$P_{v\to r}=\langle e_i,\ldots,e_j,\ldots,e_k\rangle$,其中,$e_i,e_j,e_k$等为该生成路径覆盖的有向边.生成路径的长度定义为该生成路径中有向边的数目,记为$len(P_{v\to r})$.给定生成路径的一条边e,$pre(e)$为e在$P_{v\to r}$上的前驱有向边,$succ(e)$为e在$P_{v\to r}$上的后续边.

定义 5(生成源). 给定输出对象o的一棵回溯树$t_o=(V,\chi,E)$和该树的一条生成路径$P_{v\to r}=\langle e_i,\ldots,e_j,\ldots,e_k\rangle$,则由该生成路径、起点$v$和$v$的对象集合$\chi(v)$构成的三元组$g=(P_{v\to r},v,\chi(v))$为输出对象$o$在路径$P_{v\to r}$上的生成源,$\chi(v)$为该生成源的对象集合.

在输出对象 o 的回溯树中,每一条生成路径均可实现生成源对象集合到根节点输出对象的转化.如图 3(a)所示的对象A的回溯树中,路径长度为2的生成路径 $p_{n_2\to r}=\langle e_2,e_1\rangle$ 和路径长度为3的生成路径 $p_{n_3\to r}=\langle e_3,e_2,e_1\rangle$ 分别将生成源对象集合$\{a,D,E\}$和$\{a,E,f\}$转化为输出对象 F.

性质 1. 在回溯树$t_o=(V,\chi,E)$的同一条生成路径$P_{v\to r}$上,不存在两个具有相同对象集合的节点.

证明:若该性质不成立,则可假设生成路径$P_{v\to r}$上存在两个节点v_i和v_j,满足条件$\chi(v_i)=\chi(v_j)$.由于生成路径上的每一个节点均处在回溯树的不同层,不妨设节点v_i在t_o中的层高于v_j在t_o中的层,即v_i更靠近根节点r,其关系如图 4 所示,对每一个节点v_i,其发出的有向边为e_i,e_i关联的规则为r_i,$r_{i+1}.O_d=o_i$.

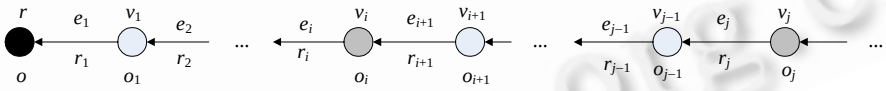

Fig.4 An example of generation path

图 4 生成路径示例

由回溯树的条件(5)可得,$o_i\notin\chi(v_{i+1})$,否则$r_{i+1}.O_s\cap r_{i+1}.O_d=o_i\neq\varnothing$与条件(5)矛盾,同理可知,$o_i\notin\chi(v_{i+2})$.假设$o_i\in\chi(v_{i+2})$,则一定存在关系$o_i\in r_{i+2}.O_d$,从而得到$r_{i+2}.O_s\cap r_{i+1}.O_d=o_i\neq\varnothing$,与条件(5)矛盾.以此类推,可知$o_i\notin\chi(v_j)$.$o_i\in\chi(v_i)\wedge o_i\notin\chi(v_j)\Rightarrow\chi(v_i)\neq\chi(v_j)$,因此,假设条件$\chi(v_i)=\chi(v_j)$不成立. □

性质 2. 在回溯树$t_o=(V,\chi,E)$的同一条生成路径$P_{v\to r}$上,不存在两条使用了相同规则的边.

证明:若该性质不成立,则可假设生成路径$P_{v\to r}$上存在两条边e_i和e_j,且$e_i.r=e_j.r=r$,不妨设e_i比e_j更靠近根节点,其关系如图 4 所示,其中,$r_i=r_j=r$,$o_{i-1}=o_{j-1}$.由性质 1 的证明过程可知,$o_{i-1}\notin\chi(v_i)$,$o_{i-1}\notin\chi(v_{i+1})$,…,$o_{i-1}\notin\chi(v_{j-1})$,然而根据假设条件$o_{i-1}=o_{j-1}$和关系$o_{j-1}\in\chi(v_{j-1})$可知,$o_{i-1}\in\chi(v_{j-1})$,因此矛盾. □

2.2 完备回溯树及其性质

对于一个输出对象,在其规则库中的回溯树并非唯一.在图 3(a)中,由节点r,n_1和n_2构成的一棵子树也是输出A的回溯树.在一个输出对象的所有回溯树中,存在一棵特别的回溯树,即完备回溯树.

定义 6(完备回溯树). 给定一个服务规则库$R_S=(I,O,R)$,一个输出对象$o\in O$,以及o的一棵回溯树$t_o=(V,\chi,E)$,对于t_o中的任意节点v,记指向v的边所关联的规则组成的集合为R^e,v的对象集合$\chi(v)$中的输出对象组成的集合为

O^v(即$O^v=\chi(v)\cap O$),规则库中目标对象集合属于集合O^v的规则组成的集合为R^v(即$R^v=\{r|r\in R,r.O_d\subseteq O^v\}$),则回溯树$t_o$为完备回溯树(记为$t_o^*$),当且仅当集合$R^{v-e}=R^v-R^e$满足以下条件之一:

(1) $R^{v-e}=\varnothing$;

(2) 对于任意规则$r\in R^{v-e}$,为t_o新增边$e=(v,v',r)$之后变成t_o',均使得t_o'因不满足回溯树条件(5)而成为非回 溯树.

上述定义表明,完备回溯树与其他回溯树相比,其任何一个节点均包含最多的儿子节点.根据定义可知,如图 3(a)所示的树为输出对象A的完备回溯树.我们以该树的节点n_3为例进行说明.对于该节点有:$R^e=\varnothing$,$O^v=\{E\}$,$R^v=\{r_4\}$,$R^{v-e}=\{r_4\}$,完备回溯树的条件(1)不满足,但条件(2)满足,这是因为对于R^{v-e}中的规则r_4,一旦为如图 3(a)所示的回溯树的节点n_3增加边e_4之后就变为如图 3(c)所示的树.在这棵树中,由于出现了对象B和E之间的循环转化(边e_4将B转化成E,在e_2中又将E转化成B),因此,它是非回溯树,即完备回溯树的条件(2)满足.通过同样的方法可以考察如图 3(a)所示的回溯树的其他节点,即可验证此回溯树为输出A的完备回溯树.

根据完备回溯树的定义,可以得到完备回溯树具备如下 3 条性质:

性质 3. 在一个输出对象的所有回溯树中,其完备回溯树包含了该输出对象的所有生成路径.

证明:假设完备回溯树不包含所有生成路径,则必定存在某条生成路径$P_{v\to r}=\langle e_i,\ldots,e_j,\ldots,e_k\rangle$不包含于完备回溯树$t_o^*$中.

1) 若$P_{v\to r}$覆盖的节点集合与t_o^*的节点集合仅有 1 个公共节点(即根节点r),则根据完备回溯树的定义,对于t_o^*中的根节点r,有$R^{v-e}\neq\varnothing$,即完备回溯树的条件(1)不满足;为t_o^*的根节点r增加生成路径p的边e_k,使得t_o^*变为另一棵回溯树$t_o^{*'}$,这与完备回溯树的条件(2)不相符,因此,t_o^*不是完备回溯树,与已知条件矛盾.

2) 若p覆盖的节点集合与完备回溯树的节点集合还含有中间公共节点(设为 n),则可采用同样的方式说明t_o^*不是完备回溯树,从而与已知条件矛盾. □

性质 4. 在一个输出对象的所有回溯树中,完备回溯树包含了该输出对象的所有生成源.

证明:假设完备回溯树不包含所有的生成源,则必存在一个生成源$\chi(v)$不属于该完备回溯树t_o^*,则该生成源的生成路径$P_{v\to r}=\langle e_i,\ldots,e_j,\ldots,e_k\rangle$也不属于$t_o^*$,这与性质 3 矛盾,因此,假设不成立. □

性质 5. 对于一个输出对象,有且仅有一棵完备回溯树.

证明:假设存在两棵不相同的完备回溯树t_o^*和$t_o^{*'}$,则t_o^*和$t_o^{*'}$一定存在以下差别之一:(1) $t_o^{*'}$(或者t_o^*)中至少存在 1 个生成源不属于t_o^*(或者$t_o^{*'}$),这与性质 4 矛盾,因此,t_o^*和$t_o^{*'}$包含相同的生成源;(2) $t_o^{*'}$(或者t_o^*)中至少存在 1 条生成路径不属于t_o^*(或者$t_o^{*'}$),这与性质 1 矛盾,因此,t_o^*和$t_o^{*'}$包含相同的生成路径.

由此可知,t_o^*和$t_o^{*'}$是由相同的生成源和生成路径构成的,即t_o^*和$t_o^{*'}$是两棵完全相同的完备回溯树,这与假设矛盾. □

3 基于回溯树的自动服务组合

给定一个服务规则库$R_S=(I,O,R)$和一个用户请求$r=\{I^r,O^r\}$,I^r为用户可以提供的输入对象集合,O^r为用户请求的目标输出对象集合.基于回溯树的服务自动组合方法分 3 步来组合服务:1) 为用户请求的每一个输出对象即时构造其完备回溯树;2) 在每一个输出对象的完备回溯树中选择一条最佳生成路径;3) 将所生成路径合成,形成大粒度流程服务.

3.1 回溯树的自动生成

给定一个服务规则库$R_S=(I,O,R)$和一个输出对象o,可通过如下CBTC算法自动创建输出o的完备回溯树$t_o^*=(V,\chi,E)$.

算法 2. CBTC (complete-backward-tree-constructor).

Input: A Service-Rule-Repository $R_S=(I,O,R)$ and an output object $o\in O$.

Output: Complete-Backward-Tree $t_o^*=(V,\chi,E)$ for o.

1. **Set** *tQueue* as a first-in-first-out queue of node;
2. **Set** *tNode* as a node;
3. **Set** the root of t_o^* as o;
4. *V.add*(*o*); //add the root node into the set *V*
5. *tQueue.push*(*o*);
6. **While** (!*tQueue.empty*()) {
7. *tNode=tQueue.top*(); //get the top node
8. *tQueue.pop*(); //eliminate the top node from the queue
9. **Set** $O^{tNode}=\chi(tNode)\cap O$;
10. **For** each $o'\in O^{tNode}$ {
11. **Set** $\mathrm{l}=\langle e_1,e_2,\ldots,e_n\rangle$ as the path from *tNode* to the root;
12. **For** each $r\in R$ {
13. **If** ($r.O_d==\{o'\}$ **&&** $r.O_s\cap\left(\bigcup_{e_i\in \mathrm{l}'} e_i.O_d\right)==\varnothing$) {
14. **Create** a new node *cNode*;
15. **Set** $\chi(cNode)=(\chi(tNode)-\{o'\})\cup r.O_s$;
16. *tQueue.push*(*cNode*);
17. *V.add*(*cNode*);
18. **Create** an edge $e=(cNode,tNode,r)$;
19. *E.add*(*e*);
20. }
21. }
22. }
23. }
24. **Return** $t_o^*=(V,\chi,E)$;

算法 CBTC 为输出对象 *o* 构造完备回溯树的过程是采用递归方式从根节点自顶向下逐层从左至右添加节点和有向边.由于规则库中的输入对象集合、输出对象集合与规则集合的元素均为有限个,且算法 CBTC 在为每一个节点添加其儿子节点时,均要求满足条件 $r.O_s\cap\left(\bigcup_{e_i\in \mathrm{l}'} e_i.O_d\right)==\varnothing$,以避免从根节点到新增节点的路径上出现对象间的循环转化而使得路径无限长,因此,在算法 CBTC 构造的回溯树中,每一条路径均是在有限步之后终止,故而算法 CBTC 总是可以在有限递归次数之后停机.

利用算法 CBTC 构造一棵包含 *n* 个节点的回溯树,假若回溯树每一个节点可包含的对象个数最多为 *m*,服务规则库中能产生同一个输出对象的规则数最多为 *q*,即回溯树的扇出为 $m\times q$,由于 CBTC 是基于树的层次遍历方式构造的,因此,我们得到其时间复杂度为 $O(n\times m\times q)$,空间复杂度为 $O(n)$.以图 2 的规则库为例,说明 CBTC 算法如何为输出 *F* 构造完备回溯树,其构造过程分如下几步完成:

(1) 初始情况为如图 5(a)所示的树,此时,*tQuque* 队列中仅有以输出对象 *F* 为标识的节点 *r*.当根节点 *r* 从队列弹出之后,得到 $tNode=r$,此时,$O^{tNode}=\{F\}$,$o'=F$.由于规则 r_5 的目标对象集合 $r_5.O_d=\{F\}$ 且 $r_5.O_s\cap\left(\bigcup_{e_i\in \mathrm{l}'} e_i.O_d\right)=$

$\{B,D,f\}\cap\varnothing=\varnothing$,因此,为节点$r$增加儿子节点$n_1$,其标识对象集合为$\chi(n_1)=\{B,D,f\}$,以及从$n_1$指向$r$的有向边$e_1=(n_1,r,r_5)$,并且将节点$n_1$加入到队列*tQuque*尾部.此时,回溯树演变为如图 5(b)所示的回溯树,CBTC的一次迭代结束.

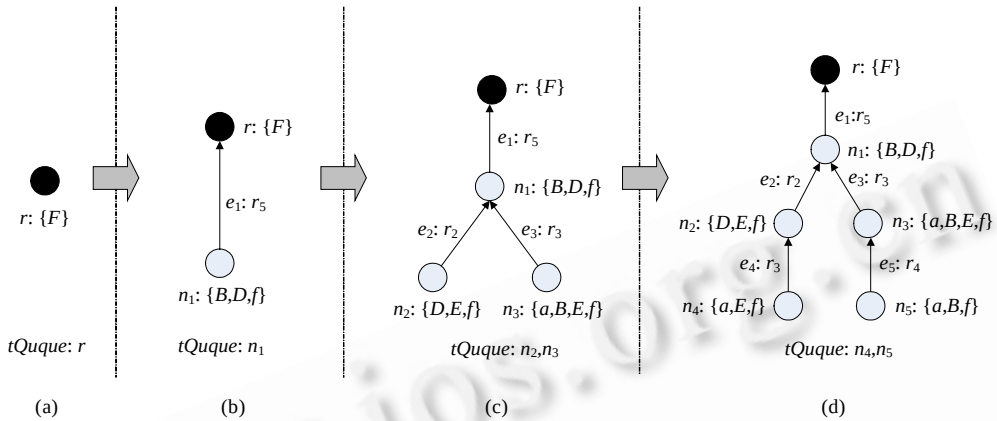

Fig.5 Construction of complete-backward-tree

图 5 完备回溯树的构造过程

(2) CBTC再次将队列*tQuque*顶部节点n_1弹出,得到$tNode=n_1$,此时,$O^{tNode}=\{B,D\}$.分别对B和D两个输出对象进行考察:① 当$o'=B$时,由于规则r_2的目标对象集合$r_2.O_d=\{B\}$,且 $r_2.O_s\cap\left(\bigcup_{e_i\in I'}e_i.O_d\right)=\{D,E\}\cap\{F\}=\varnothing$,因此,为节点$n_1$增加儿子节点$n_2$(其标识对象集合为$\chi(n_2)=\{D,E,f\}$),以及从$n_2$指向$n_1$的有向边$e_2=(n_2,n_1,r_2)$,并且将节点$n_2$加入到队列*tQuque*尾部;② 当$o'=D$时,由于规则$r_3$的目标对象集合为$r_3.O_d=\{D\}$,且 $r_3.O_s\cap\left(\bigcup_{e_i\in I'}e_i.O_d\right)=\{a,E,f\}\cap\{F\}=\varnothing$,因此,为节点$n_1$再增加一个儿子节点$n_3$(其标识对象集合为$\chi(n_3)=\{a,B,E,f\}$),以及从$n_3$指向$n_1$的有向边$e_2=(n_3,n_1,r_3)$,并且将节点$n_3$加入到队列*tQuque*尾部.

此时,回溯树如图 5(c)所示,队列*tQuque*中包含n_2和n_3两个节点.

(3) CBTC弹出节点n_2,得到$tNode=n_2$,此时,$O^{tNode}=\{D,E,f\}$,分别考察输出对象D和E:① 当$o'=D$时,采用与步骤(2)中相同的方法,可以为节点n_2添加儿子节点n_4和有向边e_4,并将n_4加入到队列尾部;② 当$o'=E$时,由于规则r_4的目标对象集合$r_4.O_d=\{E\}$,但关系 $r_4.O_s\cap\left(\bigcup_{e_i\in I'}e_i.O_d\right)=\{a,B,f\}\cap\{B,F\}=\{B\}\neq\varnothing$,因此不满足添加儿子节点的条件.

CBTC弹出节点n_3,与处理节点n_2类似,为节点n_3添加儿子节点n_5和有向边e_5,得到如图 5(d)所示的回溯树.此时,队列*tQuque*中包含n_4和n_5两个节点.

CBTC相继弹出节点n_4和n_5,并分别按照步骤(2)进行处理,但均不满足为节点n_4和n_5添加儿子节点的条件,因此算法终止,最终得到图 5(d)为输出F的完备回溯树.

3.2 生成路径的选取

对于用户请求的某个输出对象 o,利用 CBTC 构造其完备回溯树 $t_o^*=(V,\chi,E)$之后,可以得到该输出对象的所有可能的生成源和生成路径.因此,若用户期望通过规则库将其提供的输入对象集合I^r转化成目标输出对象o,则必须从o的所有生成源和生成路径中选出有效生成源和有效生成路径.

定义 7(有效生成源/有效生成路径). 给定一个用户请求$r=\{I^r,O^r\}$,一个目标输出对象$o\in O^r$,o的完备回溯树 $t_o^*=(V,\chi,E)$和 t_o^* 中的一个生成源$g=(P_{v\to r},v,\chi(v))$,若满足关系$\chi(v)\subseteq I^r$,则生成源g为用户请求r对o的一个有效生

成源,而g对应的生成路径$P_{v\to r}$为用户请求r对o的一条有效生成路径.

为了从完备回溯树中选取有效生成源(生成路径),可通过层遍历的方式访问完备回溯树中的每一个节点,判断该节点的对象集合是否包含在用户提供的输入对象集合I^r中,即判断关系$\chi(tNode)\subseteq I^r$是否满足,若满足,则该节点对应的生成源为有效生成源,该生成源的生成路径为有效生成路径.假如完备回溯树包含了n个节点,由于有效生成路径的选取是通过对该回溯树的层次遍历完成的,因此,其时间复杂度为$O(n)$,而空间复杂度为$O(1)$.每一条生成路径均可实现其生成源的对象集合到根节点输出对象的转化,但不同生成路径的生成代价不尽相同.一般而言,生成路径的路径越长,表明对象被转化的次数越多,其代价也就越大.因此,若在该对象的完备回溯树中存在多个有效生成源和有效生成路径,则将路径最短的有效生成路径作为该对象的最佳生成路径,对应的生成源作为最佳生成源.倘若在完备回溯树中不存在有效生成源(生成路径),则说明该规则库无法满足用户请求,基于回溯树的服务组合提前终止.

为了加快服务组合的速度,可以将完备回溯树的生成与最佳生成源(生成路径)的选取两个步骤结合起来.由于完备回溯树是按树的层遍历方式逐层生成,因此在每生成一个节点时,就可判断该节点是否为有效生成源,若是,则肯定为最佳有效生成源.当找到了最佳有效生成源之后,即可停止回溯树的构造.

3.3 生成路径的合成

通过回溯树的自动生成和生成路径的选取之后,用户请求的每一个输出对象都得到了一条最佳的有效生成路径,但这些路径都是彼此孤立的,因此需要将这些路径合成为一个大粒度流程服务.

定义 8(流程服务). 一个流程服务f_s是由若干操作合成,表示为一个六元组$f_s=(s,e,N,R,O_e,O_\otimes)$,其中:

s 为流程服务的入口节点,e 为流程服务的出口节点.

$N=\{n_1,n_2,\ldots,n_m\}$为流程服务中间节点集合.每一个中间节点n表示为一个三元组$n=(O_s,O_e,p)$,其中,O_s为该节点的起始对象集合,O_e为该节点的结果对象集合,p为该节点对应的操作.

$R\subseteq\{s\}\times N\cup N\times\{e\}\cup N\times N$ 为流程服务入口节点与中间节点、中间节点与出口节点,以及中间节点之间的时序关系集合.

O_e 为流程服务从入口节点s接收的外部对象集合,$O_\otimes$为流程服务从出口节点e产生的对象集合.

由于生成路径重在描述对象集合之间的转化关系,而流程服务重在刻画操作之间的时序关系,因此,为了将若干生成路径合成为流程服务,需要将生成路径转化成操作链.

定义 9(操作链). 生成路径$P_{v\to r}=\langle e_i,\ldots,e_j,\ldots,e_k\rangle$对应的操作链$q_p$为$P_{v\to r}$的边所对应的操作组成,表示为有序队列$q_p=\langle n_i,\ldots,n_j,\ldots,n_k\rangle$,队列中的每一个元素$n$与$P_{v\to r}$中的边$e$对应,可以表示为一个三元组$n=(O_s,O_e,p)$,其中,$O_s=\chi(e.v_s)$为节点的起始对象集合,$O_e=\chi(e.v_e)$为节点的结果对象集合,$p$为节点对应的操作.生成路径$P_{v\to r}$对应的生成源$g=(P_{v\to r},v,\chi(v))$的对象集合$\chi(v)$为操作链$q_p$的输入对象集合,$P_{v\to r}$的根节点$r$的对象$o$为操作链$q_p$的输出对象.

该定义表明,操作链的每一个节点与流程服务的中间节点是一致的.图 6 为生成路径$P_{v\to r}=\langle e_i,\ldots,e_j,\ldots,e_k\rangle$对应的操作链$q_p=\langle n_i,\ldots,n_j,\ldots,n_k\rangle$的示意图.对于操作链中的每一个节点$n$,其上的符号$e.r.p$表示该节点将调用的操作,节点的入线上标识了该节点的起始对象集合,出线上标识了该节点的结果对象集合.

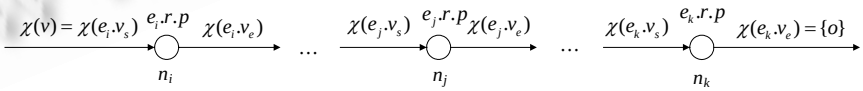

Fig.6 An example of operation-chaining

图 6 操作链示例

当用户请求$r=\{I^r,O^r\}$中指定了n个输出对象时,则根据算法OpChainComposer将这n个输出对象的最佳生成路径所对应的操作链合成为流程服务.

算法 3. OpChainComposer.

Input: $q_{p_1},q_{p_2},\ldots,q_{p_n}$: An array of Operation-Chainings.

Output: f_s=(s,e,N,R, O_e ,$O_\otimes$): A Flow-Service.

1. **Create** start node s;
2. **Create** end node e;
3. **For** each Operation-Chaining q_p=⟨n_i,…,n_j,…,n_k⟩ in the array {
4. R=R∪{(s,n_i),(n_k,e)};
5. **For** each element n in q_p in sequence {
6. N=N∪{n};
7. **If** (n≠n_k) {
8. R=R∪{(n,$succ$(n)};
9. }
10. }
11. O_e =O_e ∪n_i.O_s;
12. $O_\otimes$=$O_\otimes$∪n_k.O_e;
13. }
14. **Return** f_s;

算法 OpChainComposer 首先为流程服务创建入口节点 s 和出口节点 e,然后依次将每一条操作链置于入口节点和出口节点之间,其时间复杂度和空间复杂度均为 $O(n\times m)$,其中,n 为操作链个数,m 为操作链中包含的操作节点的最大个数.

根据算法 OpChainComposer 的合成过程,给定 n 条操作链,最后合成的流程服务的结构如图 7 所示.由于每一条操作链均由用户请求的某一输出对象的最佳有效生成路径演变过来,因此存在如下关系:

$$O_\otimes = \bigcup_{1\le m\le n} \chi(e_{km}.v_e) = O^r \tag{1}$$

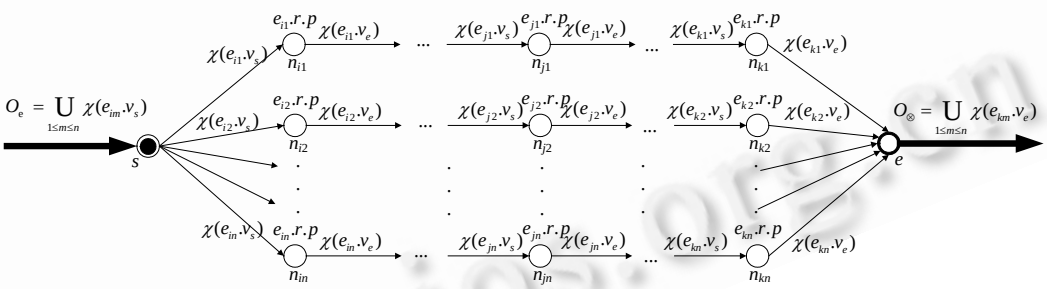

Fig.7 Flow service illustration

图 7 流程服务示意图

操作链的输入对象集合$\chi(e_i.v_s)$对应有效生成源的对象集合,对于用户请求r={I^r,O^r},关系$\chi(e_i.v_s)\subseteq I^r$成立,因此存在如下关系:

$$O_e = \bigcup_{1\le m\le n} \chi(e_{im}.v_s) \subseteq I^r \tag{2}$$

关系(1)表明,基于回溯树的服务自动组合方法得到的流程服务能够产生用户期望的所有输出对象;关系(2)则表明,该流程服务只需用户输入对象集合的一个子集作为源对象集合,就可以产生出用户期望的所有输出对象.关系(1)和关系(2)共同说明了基于回溯树的服务自动组合方法的正确性和合理性,即得到的组合服务是能够正确满足用户需求的.

3.4 自动服务组合示例

以如图 2 所示的服务规则库为例,说明如何利用基于回溯树的服务自动组合方法组合出满足用户请求

$r=\{I^r,O^r\}$(其中,$I^r=\{a,c,D,E,f\}$,$O^r=\{A,F\}$)的流程服务.根据基于回溯树的服务自动组合方法,我们分以下 3 个步骤进行服务组合:

(1) 即时构造完备回溯树

利用完备回溯树自动生成算法 CBTC 为用户请求的输出对象 A 和 F 在服务规则库中分别构造其完备回溯树 t_A^* 和 t_F^*,如图 8 所示.

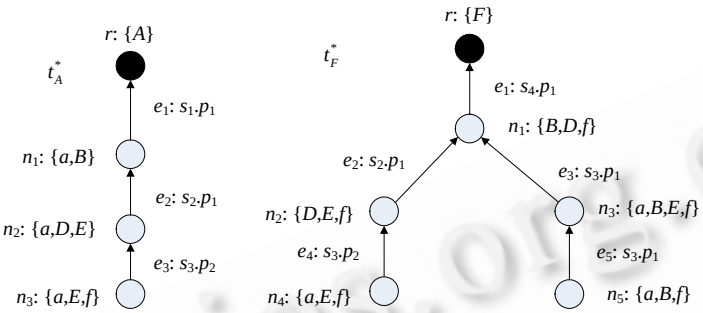

Fig.8 Complete-Backward-Trees for A and F

图 8 输出 A 和 F 的完备回溯树

(2) 选择最佳有效生成源

对输出对象 A,在 t_A^* 中存在两个有效生成源,而 $g_{n_2}=(p_{n_2\to r}=\langle n_2,n_1,r\rangle,n_2,\{a,D,E\})$ 为 r 对 A 的最佳有效生成源.对输出对象 F,在 t_F^* 中同样存在两个有效生成源,而 $g_{n_2}=(p_{n_2\to r}=\langle n_2,n_1,r\rangle,n_2,\{D,E,f\})$ 为 r 对 F 的最佳有效生成源.

(3) 合成有效生成路径

将两条最佳生成路径转变成操作链后,根据算法 OpChainComposer 将其合成为如图 9 所示的流程服务.

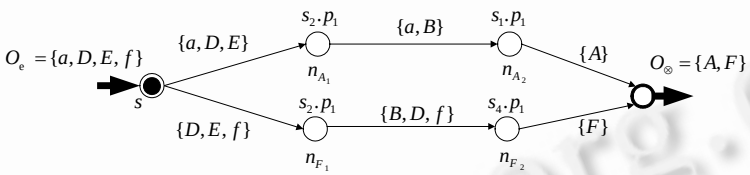

Fig.9 A flow-service constructed by the composition of two operation-chainings

图 9 由两条操作链组合而成的流程服务

该流程服务在入口节点处接收用户的输入对象集合 $O_e=\{a,D,E,f\}$ 之后,经过内部两条操作链对 3 个不同操作($s_2.p_1$,$s_1.p_1$和$s_4.p_1$)的调用后,在出口节点处产生用户期望的两个对象A和F.

4 仿真实验与结果分析

为了说明基于回溯树的服务自动组合方法的可行性和有效性,本节首先通过一组仿真实验来度量该方法在不同规模的规则库中进行服务组合时的时间开销,然后通过另一组仿真实验来度量在同一规模而具有不同对象数量的规则库中进行服务组合时的时间开销.

4.1 仿真实验准备

由于该方法建立在服务规则库的基础上,因此必须先建立服务规则库.但由于当前缺乏公共的标准服务库,我们通过模拟程序自动生成服务规则以构成规则库.在构建服务规则库$R_S=(I,O,R)$时,我们以 1~500 的自然数集合作为对象集合,每一个数字代表一个不同的对象.在为服务规则库生成一条新的服务规则$r=(O_s,O_d,p)$时,从对象集合中随机选取一个作为该规则的目标对象,从剩下的对象中随机选 1~5 个对象组成该规则的源对象集

合,并且以当前规则的序号作为该规则的标识.因此,服务规则库中的规则全部形如 {1}—r_1→{23} 或者 {4,2,5}—r_2→{8},其意义分别为规则r_1接收输入对象 1 而产生输出对象 23;规则r_2接收输入对象 4、对象 2 和对象 5 而产生输出对象 8.为方便对规则的检索,将服务规则存储于数据库中.我们采用 JDK1.5 实现基于回溯树的 Web 服务自动组合方法,在该实现中,我们将回溯树的即时构造与生成路径的选取两个步骤合并.仿真实验运行在 IBM X260 服务器上,其软硬件配置为 1.86G Xeon CPU,G RAM,Redhat Linux Operating System, MySQL 5.0.

4.2 在不同规模的规则库中进行仿真实验

采用上述服务规则的模拟生成方法,我们产生 6 个不同规模的规则库R_{s1},R_{s2},…,R_{s6},使得它们分别含有 100,200,500,1 000,2 000 和 5 000 条不同的服务规则.为了使对象的个数与规则数的比例保持一致,我们在生成这 6 个服务规则库时,分别从 500 个对象中选出 10,20,50,100,200 和 500 个对象作为对象集合.在每一个规则库中依次提交 100 次不同的用户请求,对于每一次用户请求$r=\{I^r,O^r\}$,其构造过程为:随机从对象集合中选取 3~5 个对象组成用户的输入对象集合I^r,然后,从剩下的对象中随机选取 1~3 个对象组成用户期望得到的输出对象集合O^r.

表 1 显示的是该自动服务组合方法在为 100 个用户请求组合服务后的结果.该结果表明,规则库中的规则数越多,用户请求被满足的可能性也越大,这也与实际情况相符.我们记录每一次用户请求被处理的时间,最后分别计算可满足的与不能满足的两种用户请求在不同规则库中的平均处理时间,如在规则库R_{s1}中进行的 100 次服务组合,我们计算其中 39 次可满足的用户请求的平均处理时间和 61 次未能满足的用户请求的平均处理时间,结果如图 10 所示.

Table 1 Results of requests in repositories with different scales

表 1 用户请求在不同规模的规则库中的处理结果

Rule repository	Number of rules	Number of objects	Object set	Number of satisfiable user requests	Number of unsatisfiable user requests
R_{s1}	100	10	Natural number 1~10	39	61
R_{s2}	200	20	Natural number 1~20	47	53
R_{s3}	500	50	Natural number 1~50	51	49
R_{s4}	1 000	100	Natural number 1~100	79	21
R_{s5}	2 000	200	Natural number 1~200	83	17
R_{s6}	5 000	500	Natural number 1~500	91	9

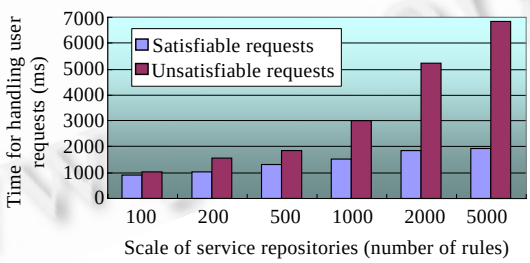

Fig.10 Time comparison in service repositories with different scales

图 10 不同规模规则库中的时间比较

图 10 显示的结果也表明,在同一个规则库中,对于无法满足的用户请求,其处理时间比可满足的用户请求的处理时间要长,这是因为对于无法满足的用户请求,都必须为用户请求的每一个输出对象建立其完整的完备回溯树才能判定该回溯树中不存在有效生成源(生成路径).此外,该结果也表明,对于无法满足的用户请求,随着规则库中规则数的增加,其处理时间也呈明显上升趋势,但趋势较为平缓.

因完备回溯树的即时构建是该自动服务组合方法的关键步骤,为此,我们在每一个规则库中随机选取 10 个不同对象,利用完备回溯树的自动生成算法 CBTC 分别为这 10 个对象即时构建完备回溯树,记录每棵完备回溯

树的生成时间,计算 10 棵回溯树的平均生成时间,结果如图 11 所示.结果表明,在不同规模的规则库中构建对象的完备回溯树的时间开销随着规则库规模的增大而有所递增,但其增长趋势较为平缓.这与图 10 中对于无法满足的用户请求的平均处理时间的增长趋势保持一致,这是因为对于无法满足的用户请求,其处理时间包含了多次完备回溯树的构建过程,因此两者的增长趋势基本相同.

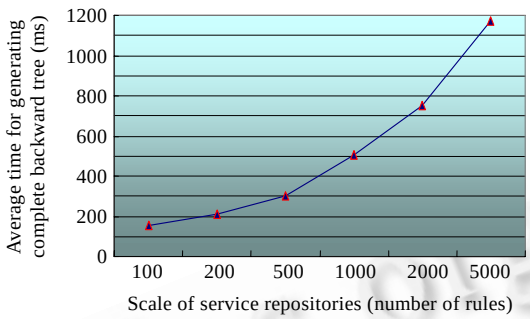

Fig.11 Time comparison for generating complete-backward-tree in service repositories with different scales

图 11 在不同规模的规则库中生成完备回溯树的时间比较

4.3 在相同规模而具有不同对象数量的规则库中进行仿真实验

在上一节的实验中,规则库的规模不等,但对象与规则数的比例相同.为了在相同规模而具有不同对象数量的规则库中进行实验,我们产生 6 个均包含 1 000 条规则的规则库R_{s1},R_{s2},…,R_{s6},使得它们的对象集合分别含有 50,100,200,300,400 和 500 个对象.与上一组实验方法相同,我们对每一个服务库提交 100 个用户请求,得到如表 2 所示的处理结果.该结果表明,在服务规则库中的规则数相同的情况下,尽管对象集合中的对象个数不同,但可满足的用户请求数没有呈现出明显的变化趋势,而基本上保持一致.这说明该方法与服务规则库中的对象数量没有直接关系,这与实际情况也相符.同样,我们得到每一个规则库中可满足的用户请求和无法满足的用户请求的平均处理时间,结果如图 12 所示.该结果表明,可满足的请求的处理时间比无法满足的请求的处理时间要少,这与第 1 组实验结果一致.该结果也表明,对于可满足的请求而言,其处理时间没有明显的波动,而基本保持相同.同样,对于不可满足的请求也是如此.由此可知,基于回溯树的服务自动组合方法的性能受服务库中的对象个数的影响较小.

以上两组实验结果均表明,基于回溯树的服务自动组合方法无论对于可满足的用户请求还是无法满足的用户请求,均能在短时间内作出反馈,特别是对可满足的用户请求,能快速组合出可执行的流程服务以满足用户需求.

Table 2 Results of requests in repositories with the same scale but different numbers of objects

表 2 用户请求在相同规模而具有不同对象数量的规则库中的处理结果

Rule repository	Number of rules	Number of objects	Object set	Number of satisfiable user requests	Number of unsatisfiable user requests
R_{s1}	1 000	50	Natural number 1~50	80	20
R_{s2}	1 000	100	Natural number 1~100	79	21
R_{s3}	1 000	200	Natural number 1~200	82	18
R_{s4}	1 000	300	Natural number 1~300	77	23
R_{s5}	1 000	400	Natural number 1~400	79	21
R_{s6}	1 000	500	Natural number 1~500	81	19

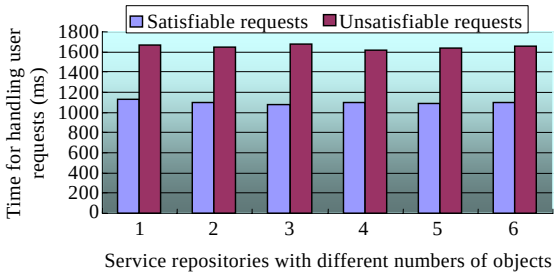

Fig.12 Time comparison in repositories with different numbers of objects

图 12 在具有不同对象个数的规则库中的时间比较

5 结束语

Web 服务组合是面向服务的计算(SOC)中的一个关键问题.本文提出了一种基于回溯树的服务自动组合方法.该方法将服务进行规则化表达之后,将服务库演变成规则库,将不同功能的服务统一成规则,强调了对象间的转化关系.在进行自动服务组合的过程中,采用分步分治的思想,首先对用户请求的每一个输出对象单独生成完备回溯树,在完备回溯树的即时构造过程中,选取输出对象的最佳生成路径和生成源,然后将生成路径合成为大粒度的流程服务以满足用户请求.该方法属于基于图搜索的自动服务组合方法的范畴,与已有的方法相比,该方法具有搜索空间小、无循环搜索、复杂度低和速度快的特点.仿真实验结果表明,该方法能够在短时间内实现服务的自动组合,从而完成用户请求.

今后的工作包含 3 个方面:1) 如何将组合之后的流程服务进行规约化简,这一点可以在工作流建模理论的基础上展开;2) 在为用户请求的目标对象选取最佳生成路径和生成源时,将服务的负载均衡问题纳入考虑范围,也就是说,既考虑局部生成路径的长度,又考虑全局服务流程中所调用的服务(操作)的负载均衡,文献[17]为这方面的工作提供了较好的参考;3) 在规则库中考虑对象之间的语义转化,挖掘更多的对象转化规则,以提高服务组合的成功率.对象的语义相似度计算的研究成果可作为这些工作的基础.

References:

[1] Yue K, Wang XL, Zhou AY. Underlying techniques for Web services: A survey. Journal of Software, 2004,15(3):428−442 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/15/428.htm

[2] Papazoglou MP, Georgakopoulos D. Service oriented computing. Communications of the ACM, 2003,46(10):25−28.

[3] Dustdar S, Schreiner W, Schreiner W. A survey on Web services composition. Int'l Journal of Web and Grid Services, 2005,1(1): 1−30.

[4] Milanovic N, Malek M. Current solutions for Web service composition. IEEE Internet Computing, 2004,8(6):51−59.

[5] Cardoso J, Sheth A. Semantic e-workflow composition. Journal of Intelligent Information System, 2003,12(3):191−225.

[6] Hamadi R, Benatallah B. A Petri net-based model for Web service composition. In: Schewe KD, ed. Proc. of the 14th Australasian Database Conf. Adelaide: Australian Computer Society, 2003. 191−200.

[7] Hu HT, Li G, Han YB. An approach to business-user-oriented larger-granularity service composition. Chinese Journal of Computers, 2005,28(4):694−703 (in Chinese with English abstract).

[8] Wu D, Parsia B, Sirin E, Hendler J, Nau D. Automating DAML-S Web services composition using SHOP2. In: Fensel D, ed. Proc. of the Int'l Semantic Web Conf. Sanibel Island: Springer-Verlag, 2003. 250−262.

[9] Rao JH, Kungas P, Matskin M. Logic-Based Web services composition: from service description to process model. In: Zhang LJ, ed. Proc. of the Int'l Conf. on Web Services. San Diego: IEEE Computer Society, 2004. 446−453.

[10] Akkiraju R, Srivastava B, Ivan AA, Goodwin R, Syeda-Mahmood T. SEMAPLAN: Combining planning with semantic matching to achieve Web service composition. In: Leymann F, ed. Proc. of the Int'l Conf. on Web Services. Chicago: IEEE Computer Society, 2006. 37−44.

[11] Zhao HB, Doshi P. A hierarchical framework for composing nested Web processes. In: Dan A, ed. Proc. of the Int'l Conf. on Service-Oriented Computing. Chicago: Springer-Verlag, 2006. 116−128.

[12] Zhang RY, Arpinar BI, Aleman-Meza B. Automatic composition of semantic Web services. In: Zhang LJ, ed. Proc. of the Int'l Conf. on Web Services. Las Vegas: IEEE Computer Society, 2003. 38−41.

[13] Aversano L, Canfora G, Ciampi A. An algorithm for Web service discovery through their composition. In: Zhang LJ, ed. Proc. of the Int'l Conf. on Web Services. San Diego: IEEE Computer Society, 2004. 12−19.

[14] Brogi A, Corfini A, Popescu R. Composition-Oriented service discovery. In: Gschwind T, ed. Proc. of the Int'l Conf. on Software Composition. Edinburgh: Springer-Verlag, 2005. 15−30.

[15] Liu JM, Gu N, Shi BL. Non-Backtrace backward chaining dynamic composition of Web services based on mediator. Journal of Computer Research and Development, 2005,42(7):1153−1158 (in Chinese with English abstract).

[16] Hashemian SV, Mavaddat F. A graph-based framework for composition of stateless Web services. In: Bernstein A, ed. Proc. of the European Conf. on Web Services. Zürich: IEEE Computer Society, 2006. 75−86.

[17] Li WZ, Guo S, Xu P, Lu SL, Chen DX. An adaptive load balancing algorithm for service composition. Journal of Software, 2006, 17(5):1068−1077 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/17/1068.htm

附中文参考文献:

[1] 岳昆,王晓玲,周傲英.Web 服务核心支撑技术:研究综述.软件学报,2004,15(3):428−442. http://www.jos.org.cn/1000-9825/15/428.htm

[7] 胡海涛,李刚,韩燕波.一种面向业务用户的大粒度服务组合方法.计算机学报,2005,28(4):694−703.

[15] 刘家茂,顾宁,施伯乐.基于 Mediator 的 Web Services 无回溯反向链动态合成.计算机研究与发展,2005,42(7):1153−1158.

[17] 李文中,郭胜,许平,陆桑璐,陈道蓄.服务组合中一种自适应的负载均衡算法.软件学报,2006,17(5):1068−1077. http://www.jos.org.cn/1000-9825/17/1068.htm

邓水光(1979－),男,浙江杭州人,博士,CCF 学生会员,主要研究领域为流程管理,面向服务的计算,社会计算.

吴健(1976－),男,博士,副教授,CCF 高级会员,主要研究领域为 Web 服务,网格计算,数据挖掘.

李莹(1973－),男,博士,副教授,主要研究领域为中间件技术,软件体系架构,编译技术.

吴朝晖(1966－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为网格计算,嵌入式计算,普适计算.