

一种执行代价较小的 Web 服务动态合成方法

轩爱成, 骆翔宇, 沙宗鲁, 董荣胜

(桂林电子科技大学 计算机与控制学院, 广西 桂林 541004)

摘 要: 在 Web 服务无回溯反向链动态框架基础上, 提出一个较小执行代价的反向链动态合成方法. 该算法根据用户的输入、输出和 Web 服务的执行代价, 在合成过程中的每一步都找到执行代价最小的 Web 服务, 从而动态生成一个执行代价较小的合成方案. 该合成方案考虑了 Web 服务的合成效率和执行代价, 特别适合于对两者都有要求的应用领域.

关键词: Web 服务合成; 执行代价; 反向链; 产生式模型

中图分类号: TP393

文献标识码: A

文章编号: 1000-7180(2011)02-0166-05

A Dynamic Web Services Composition Method with Less Execution Price

XUAN Ai-cheng, LUO Xiang-yu, SHA Zong-lu, DONG Rong-sheng

(School of Computer and Control, Guilin University of Electronic Technology, Guilin 541004, China)

Abstract: Based on non-backtrace backward chaining Web Services dynamic composition framework, this paper proposes a backward chaining dynamic composition method to lower the execution price. In order to generate this synthesis program dynamically, the Web Service with minimal execution price is found in each step of the composition course of the algorithm based on user input/output and the execution price of Web Service. The composition approach takes into account the efficiency and execution price, so it is especially suitable for users who have requirements on the both.

Key words: web services composition; execution price; backward chaining; rule model

1 引言

当前, 网络在人们工作和生活中发挥着巨大的作用. Web 服务技术为分布式异构的空间信息共享和互操作提供了有效的框架, 愈来愈多的空间信息服务以 Web 服务的形式发布出来. Web 服务技术的广泛接受使得 Web 服务正逐步成为 Internet 网络环境中资源封装的标准形式. 尽管 Web 服务技术给企业之间和企业内的应用程序的集成带来了方便, 但是单一的 Web 服务功能简单、有限, 难以满足某些实际应用的需要, 因此有必要对现有的单个 Web 服务进行合并, 以生成功能更复杂、更强大的

Web 服务来支持各种应用需求.

2 Web 服务的合成方法

Web 服务合成就是通过服务查找以及服务之间的接口集成, 将多个自治的 Web 服务根据应用需求进行组合, 从而提供新的、功能更强的 Web 服务, 或者说提供一些增值的 Web 服务. 由于当前尚未有统一的定义, 通常认为 Web 服务合成是为了达到用户特定目标, 在给定的一些 Web 服务中, 找到一个能满足这一目标的合成 Web 服务.

Web 服务合成的方式一般可抽象地划分为静态合成与动态合成两种, 其主要区别在于选择被组

收稿日期: 2010-01-13; **修回日期:** 2010-04-05

基金项目: 国家自然科学基金项目(60763004); 中国博士后科学基金(20090450389); 广西自然科学基金(0991242); 广西青年科学基金(桂科青 0728090); 广西研究生教育创新计划项目(2008105950812M424)

合服务的时机不同,前者是由设计人员在设计时确定,而后者则是在运行时选择.采用静态合成还是动态合成是根据Web服务自身的特点和用户的需求决定的.

根据不同标准和具体需求,人们从不同的角度对Web服务的合成方式进一步分类.文献[1]将目前提出的许多Web服务合成方法,大体上分成基于工作流的合成和基于语义的合成.基于工作流的合成是通过定义一个Web服务的执行流程来实现的.基于语义的合成,是从语义Web上发展起来的基于语义的Web服务合成方法.文献[2]将Web服务组合分为主动式/反应式的服务组合和强制式/选择式的服务组合(mandatory/ optional composition)两大类.其中主动式服务组合是一种离线式或预编译的稳定的组合方式;而反应式服务组合则是一种在线的、动态的和实时的组合方式.文献[3]建立了一种融合上述多种组合方式优点、基于案例库推理的Web服务组合模型.当然,Web服务合成还存在许多其他的合成方法^[4]这里就不再详述.近年来,随着“Web服务”成为开放网络环境下资源封装与抽象的核心概念,通过动态地组合服务实现资源的灵活聚合成为技术发展的自然思路.

3 执行代价

在Web环境下,可能有多个Web服务都提供相同的功能,但是其执行代价、可靠性、和可用性等不一定相同.本文根据Web服务的执行代价自动合成Web服务.

执行代价^[5]是指一个Web服务某一个操作的代价,不同的操作有不同的执行代价.一个Web服务的执行代价 $Q_{\text{cost}}(\text{WS})$ 是服务请求者调用执行该Web服务所应当支付的费用.服务提供者或者直接发布它们的Web服务的执行代价,或者提供某种手段使得服务请求者可以获取Web服务的执行代价.一个合成的Web服务如果需要调用 n 个Web服务: $\text{WS}_1, \text{WS}_2, \dots, \text{WS}_n$ 这 n 个Web服务中可以有相同的Web服务,因为有时一个合成的Web服务有可能重复调用某些Web服务,那么它的总执行代价是:

$$Q_{\text{cost}}(\text{WS}) = Q_{\text{cost}}(\text{WS}_1) + \dots + Q_{\text{cost}}(\text{WS}_n)$$

4 较小执行代价动态反向链合成

4.1 较小执行代价动态合成方案框架

文中提出一种动态组合算法,并套用了文献[1]

中Web服务无回溯反向链组合框架及其执行引擎.该方法首先对Web服务进行建模,并利用一个本体来消除合成过程中可能出现的语义冲突,然后采用合成算法根据用户的输入、所期望的输出和Web服务本身的执行代价动态生成合成方案,最后可把该合成方案自动生成一个合成Web服务.

其生成合成方案的框架如图1所示.

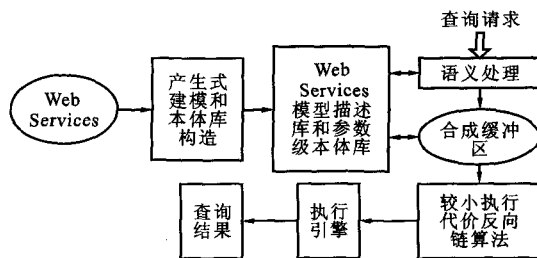


图1 Web服务较小执行代价动态合成框架

整个合成步骤如下:

(1) 建模与本体库构造:合成之前首先对Web服务进行产生式建模并放入Web服务模型描述库中;同时还需要构造一个本体库来描述模型之间的语义关系.本体构造在本体中定义领域术语、Web服务输入输出参数之间的语义关系,包括同名异义和异名同义等.

(2) 语义处理:根据模型描述库和本体库,把某些模型的输入输出替换为领域术语,解决合成过程中可能碰到的语义冲突问题,并重新放入合成缓冲区,以加快合成效率.

(3) 算法合成:利用较小执行代价反向链算法对缓冲区中的模型进行合成.

(4) 转换执行:转换合成方案、执行并返回查询结果.合成算法生成的一个合成方案可以自动转换为一个合成的Web服务,然后可以使用引擎BP-WS4J^[6]来执行并返回结果.

合成缓冲区用来存放经过语义冲突处理后的、根据输出参数合成方案要调用Web服务的产生式模型,并在该缓冲区完成从功能相同的Web服务模型中选出执行代价最小的服务.Web服务模型放在合成缓冲区,一方面是为了不改变模型库中模型本来形式;另一方面是为了选出代价最小的Web服务提供一定的内存区域.

4.2 产生式建模及参数级本体构造

文中通过文献[1,7]中产生式的形式来对Web服务进行建模,这种模型是把一个Web服务表示为一个产生式,产生式的体部和头部分别表示Web服务的输入和输出,且该产生式的头部只有一项.如果

Web 服务有多个输出,则需要建模成多个产生式.并且,该 Web 服务有自己的执行代价 cost. 由于这种建模采用产生式的形式,能够清晰直观地区分输入和输出.

假定一个 Web 服务要求 m 个输入: $I_1, I_2, \dots, I_m$, 并产生 n 个输出: $O_1, O_2, \dots, O_n$, 那么它就可以表示为 n 个产生式(假定每一个 Web 服务只有一个 Operation):

$$I_1 \& I_2 \& \dots \& I_m \rightarrow O_1$$

$$I_1 \& I_2 \& \dots \& I_m \rightarrow O_2$$

...

$$I_1 \& I_2 \& \dots \& I_m \rightarrow O_m$$

也就是说,对任何一个 Web 服务,都可以表示为一个或者多个二元组的形式 $S(I, o)$. 其中 I 是输入集合, o 为单个输出. 另外,文中对以下两种特殊情况不予考虑:

(1) 当 $m = 0$ 时,这个 Web 服务不需要输入数据,它表示所有的数据都可以作为其输入,而不是不能有数据输入.

(2) 当 $n = 0$ 时,这样的 Web 服务对合成来说没有任何意义,它大多是用来传递控制信息的.

由于 Web 服务的实现平台可能不同,且来自不同的领域,对它们进行合成时,难免会碰到一些语义上的问题. 为了解决这个问题,文中引用文献[1]提出一种参数级本体(Parameter Level Ontology)来描述 Web 服务的输入参数、输出参数(即产生式的体部项和头部项)以及领域模型的参数(即领域术语)之间的概念关系.

假定在股票行业中有这样一个情景,用户想通过一个“跨国公司名字”和“某一特定时刻”,找出该跨国公司在某一股市的股票折合成人民币后的价格以及该公司所在国家名称和国家其它信息. 不妨假定有以下 7 个 Web 服务:

① Country1 (CountryCodeⁱ, CountryName^o, Info^o, 0.4)

② Country2 (CountryCodeⁱ, CountryName^o, Info^o, 0.2)

③ UStoRMB (USPriceⁱ, Datetimeⁱ, RMBPrice^o, 0.5)

④ UKtoRMB (UKPriceⁱ, GivenDatetimeⁱ, RMBPrice^o, 0.6)

⑤ New YorkStock1 (CompanyIDⁱ, SomeDatetimeⁱ, USPrice^o, 0.7)

⑥ New YorkStock2 (CompanyIDⁱ, SomeDate-

timeⁱ, USPrice^o, 0.5)

⑦ YellowPage (CompanyNameⁱ, CompanyID^o, Address^o, State^o, CountryID^o, Zip^o, Phone^o, info^o, 0.8)

可以把 Web 服务写成带有绑定模式的谓词形式,谓词中的属性的上标 i 表示 Web 服务的输入参数,上标 o 表示该 Web 服务的输出结果,最后一项为该 Web 服务的执行代价.

根据产生式建模和参数级本体构造方法,构造的 Web 服务产生式模型如表 1 所示.

表 1 消除语义冲突后的 Web 服务产生式模型

编号	产生式	执行代价	服务
1	CountryID ⁱ →CountryName	0.4	①
2	CountryID ⁱ →CountryInfo	0.4	①
3	CountryID ⁱ →CountryName	0.2	②
4	CountryID ⁱ →CountryInfo	0.2	②
5	USPrice ⁱ &Datetime ⁱ →RMBPrice	0.5	③
6	UKPrice ⁱ &Datetime ⁱ →RMBPrice	0.6	④
7	CompanyID ⁱ &Datetime ⁱ →USPrice	0.7	⑤
8	CompanyID ⁱ &Datetime ⁱ →USPrice	0.5	⑥
9	CompanyName ⁱ →CompanyID	0.8	⑦
10	CompanyName ⁱ →Address	0.8	⑦
11	CompanyName ⁱ →CountryID	0.8	⑦
12	CompanyName ⁱ →State	0.8	⑦
13	CompanyName ⁱ →Zip	0.8	⑦
14	CompanyName ⁱ →Phone	0.8	⑦
15	CompanyName ⁱ →CompanyInfo	0.8	⑦

4.3 较小执行代价反向链算法

该算法从用户所需的查询结果开始,试图找出能得到该结果的所有产生式,并从其中选出执行代价最小的产生式. 根据产生式,找出对应的 Web 服务将其加入到合成方案中.

4.3.1 算法描述

UI: 用户的输入;

UO: 用户要求的输出;

OT: 所有可用的 Web 服务的对应的输出集合;

IT: 所有可用的 Web 服务的对应的输入集合;

RS: 所有产生式的集合;

WSU: 合成方案中用到的 Web 服务集合;

RU: 合成方案中用到的产生式集合;

RZ: 产生式暂存的集合;

OA: 输出目标;

OC: 输出目标;

(1) if UI 用户的输入不完全包含在 IT 中 then

返回用户的输入要求错误;

(2) if UO 用户的输出不完全包含在 OT 中
then 返回用户的输出要求无法实现;

(3) OA=UO;

(4) while(OA 非空)do

{

(5) 对 OA 中的一个目标 o 从 RS 中找出所有的产生式 $r: p_1, \dots, p_n \rightarrow o$ 并将所有产生式放到 RZ 中;

(6) 从 RZ 中找到执行代价最小的产生式 $r_s: p_1, \dots, p_n \rightarrow o$;

(7) if r_s 不在 RU 中 then 将 r_s 放到 RU 中;

(8) if r_s 中的 $p_1, \dots, p_n$ 不在 OC 中 then 将其加入 OC;

(9) if r_s 对应的 Web 服务不在 WSU 中 then 将其加入 WSU 中;

(10) 从 OA 中去除 o;

}

(11) OA=OC;

(12) OC 初始化为空;

(13) if UI 完全包含在 OA 中 then 输出 ok, RU, WSU;

else go to (4).

4.3.2 实例验证

通过前面的实例来展示该算法的实现过程.

用户要求的输出结果:

UO = {RMBprice, CountryName, CountryInfo};

用户的输入信息:

UI = {CompanyName, SomeDateTime};

OT = {Web Services①~⑦对应的输出};

IT = {Web Services①~⑦对应的输入};

RS = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15}; 将 UO 赋给 OA;

OA = {RMBprice, CountryName, CountryInfo};

当 OA 非空时从 OA 中取出 o=RMBprice;
依次执行将会得到:

RZ = {5, 6};

$r_s = 5$;

RU = {5};

OC = {USprice, Datetime};

WSU = {③};

从 OA 中去除 RMBprice,

OA = {CountryName, CountryInfo};

同理, 可以计算 o=CountryName, o=CountryInfo 对应 RZ, RU, OC, WSU 的集合;

当 OA 为空时, 将集合 OC 赋给 OA 重复上面的 while 操作. 直到 UI 完全包含在 OA 中.

在调用该算法结束后有如下结果:

UI = {CompanyName, SomeDateTime};

UO = {RMBprice, CountryName, CountryInfo};

RS = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15};

RU = {3, 4, 5, 8, 9, 11};

为满足用户需求选出的 Web 服务 WSU = {②, ③, ⑥, ⑦}.

5 合成方案性能分析

首先该合成方案中不包含多余的 Web 服务, 并且语义处理不会改变模型描述库中模型本来的形式, 因此不会影响下次的合成; 在合成方案动态的生成过程中, 每一步都找到执行代价最小的 Web 服务, 因此最后的合成方案的执行代价会比较小. 如果用户所要求的输入和输出结果只需要一步调用就可以完成, 那么该合成方案就是执行代价最小的合成方案.

从合成过程看, 文中算法、无回溯反向链算法和最小执行代价算法^[5]在时间效率上主要表现如下:

(1) 文中算法的合成时间: 语义处理时间 + 较小执行代价反向链算法合成时间.

(2) 无回溯反向链合成时间: 语义处理时间 + 无回溯反向链算法合成时间 + 输入闭包的计算时间.

(3) 最小执行代价合成时间是所有合成方案的时间总和, 而每个方案的合成时间为: 语义处理时间 + 无回溯反向链算法合成时间 + 输入闭包的计算时间.

由于文中算法的执行时间与无回溯反向链算法相当, 所以比无回溯反向链合成时间少了输入闭包的计算时间. 明显看出该合成方案比最小执行代价自动合成节省了许多时间. 当 Web 服务规模比较大时该合成方案的优势更明显.

6 结束语

文中提出了较小执行代价反向链算法. 该算法在不改变模型库中模型的基础上, 针对每一个输出,

在功能相同的 Web 服务中动态的选择执行代价最小的一个. 该算法在兼顾 Web 服务执行代价的同时, 实现了 Web 服务的动态快速组合.

参考文献:

- [1] 刘家茂. Web Services 动态合成及 UDDI 注册/查询技术的研究 [D]. 上海: 上海复旦大学, 2005.
- [2] Dipanjan Chakraborty, Anupam Joshi. Dynamic service composition: state-of-the-art and research directions [R]. University of Maryland, Department of Computer Science and Electrical Engineering, USA, 2001.
- [3] Benchaphon Limthanmaphon, Yanchun Zhang. Web service composition with case based reasoning [C]// Proc. of the 14th Australian Databases Conference. [S. l]: [s. n], 2003(17): 201-208.
- [4] 刘青, 连祥鉴. 基于上下文的 Web 服务动态组合[J]. 微电子学与计算机, 2008, 25(9): 50-52.
- [5] Jiamao Liu, Chenhui Fan, Ning Gu. Web services automatic composition with minimal execution price [C] // Proc. of 2005 IEEE International Conference on Web Services (ICWS2005). [S. l]: [s. n], 2005: 302-309.
- [6] IBM. IBM business process execution language for Web Service java run time (BPWS4J) [EB/OL]. [2009-06-08]. <http://www.alphaworks.ibm.com/tech/bpws4j>.
- [7] Joseph C Giarratano, Gary D Riley. Expert system principles and programming [M]. 3rd ed. Beijing: China Machine Press, 2003: 143-148.

作者简介:

轩爱成 女, (1982-), 硕士研究生. 研究方向为模型检测、Web 服务.

骆翔宇 男, (1974-), 博士, 副教授. 研究方向为模型检测、Web 服务.

沙宗鲁 男, (1981-), 硕士研究生. 研究方向为信息安全技术.

董荣胜 男, (1965-), 教授. 研究方向为网络安全协议、形式化技术、计算思维与计算机方法论.