基于Dijkstra算法的Web服务合成选择策略研究

刘棱鹏,夏元友,姜建华
(武汉理工大学 计算机科学与技术学院,湖北 武汉 430070)

摘 要:Web服务合成中操作的选择是一个关键问题,这直接影响到用户对合成的复合服务的满意度,解决该问题的关键是对候选Web服务的输入输出数据关系进行建模,以及有效利用这些已有的数据依赖关系实现服务合成的请求。通过从Web服务规范语言中提取Web服务的语义信息,构建Web服务的有向图,并分析Dijkstra算法用于Web服务合成的问题。提出了相应的解决办法,给出了一个基于Dijkstra算法的Web服务合成选择策略的算法。该算法能在合成中选择最恰当的操作组合,产生最终的复合服务。

关键词:Web服务;Web服务合成;依赖图;选择策略
中图分类号:TP301.6 **文献标识码**:A **文章编号**:1673-629X(2008)02-0104-04

Dijkstra Algorithm-Based Selection Strategy in Web Services Composition

LIU Ling-wei, XIA Yuan-you, JIANG Jian-hua
(College of Computer Science and Technology, Wuhan University of Technology, Wuhan 430070, China)

Abstract: The choice of operation is a key problem in the Web services composition. It influences customer's satisfaction to the quality of the Web service composition directly. The key which solve this problem is to construct a model for input and output data dependency relationships of candidate Web services and realize the Web service composition request by using this available data dependency relationships. Using the semantic information which is abstracted from Web services specification languages, a so-called dependency graph is constructed, and analyze the problem which turn out when Dijkstra algorithm used in the Web services composition. Then an algorithm based on Dijkstra algorithm for the choice strategy in Web services composition is designed. This algorithm can choose the most fitting operation in the Web services composition and compound the end complex service.

Key words: Web services; Web services composition; dependency graph; selection strategy

0 引 言

Web服务的出现使互联网的应用模式发生了巨大的改变,它已成为现代互联网技术发展的趋势。Web服务提供了从简单请求到复杂商务处理的系列功能,一旦Web服务被部署,其他的应用包括Web服务本身就能够发现并调用所部署的服务。因此,越来越多的企业将自己的业务作为Web服务发布。然而,一个单独的Web服务很可能限制其所拥有的能力。所以工业界和学术界都希望能够通过合成现有的Web服务来创造新的Web服务,服务的合成也因此成为当前研究的热点。

现在已经出现了许多支持Web服务合成的系统,如由HP实验室开发的电子商务服务合成系统e-Flow[1],基于Agent的服务发现的Web服务合成系统AgGlow[2],采用对等服务协调模型的服务合成系统Self-Serv[3]。但是目前许多系统没有使用选择策略,在合成的过程中只是在可能的方案中随机选择几个服务进行合成,或者产生所有可能的合成方案,然后用户进行二次选择。随着技术的不断发展,人们开始对合成的质量提出了较高的要求,很多情况下人们希望根据合成的代价选择相应的服务进行合成[4],比如要求选择的服务合成的资源开销最小,或者选择的服务合成的时间最短等,这样的要求上述服务合成系统显然已经不能胜任了,有的系统在用户的参与下勉强还可以得出结果,但是效率和准确度难以保证,有的系统完全就无法完成这样的合成。

针对目前服务合成系统的不足,文中将经典的Dijkstra算法应用到Web服务合成的选择中,这样就使

收稿日期:2007-05-22
基金项目:国家自然科学重点基金项目(E055033502)
作者简介:刘 (1983-),男,湖北远安人,硕士研究生,研究领域为计算机智能技术、计算机网络应用;夏元友,博士,教授,博导,研究领域为计算机智能技术与智能系统、计算机网络应用技术。

得服务合成系统能根据用户的要求来选择最恰当的服务进行合成。

1　问题背景

1.1　Web 服务与 Web 服务合成

Web 服务可以看成是部署在 Internet 上的 API，它可以方便被应用程序甚至其它 Web 服务集成和调用，形成新的应用服务。它具有完好的封装性、松散耦合、高度可集成能力。毫无疑问，Web 服务技术将成为下一代 Web 的主流技术，它是实现“软件作为一种服务”的体现。

Web 服务的体系结构如图 1 所示，由服务请求者、服务代理者和服务提供者组成。

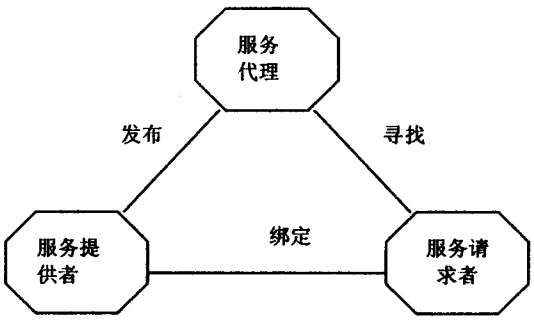

图 1　Web 服务体系的基本结构

所谓 Web 服务的合成，指的是从互联网中选取相对简单、可用的 Web 服务并将它们组合成新服务的技术。合成后的新服务被称为复合服务；用于合成复合服务的子服务称之为构件服务。Web 服务合成大致可以分为两种类型：静态合成和动态合成。在设计阶段就定义了复合服务规范的合成方法是静态合成；相对地，如果在运行时刻所需服务才被选择和调用的服务合成方法属于动态合成。在这里要强调的是，服务的合成尤其是动态合成是一个非常复杂的过程，涉及到许多的难题。文中主要研究的是静态合成，针对合成中服务的选择策略进行研究并提出了一种可行的方案。

1.2　Web 服务合成模型表示

为了配合算法的提出，首先要对 Web 服务合成进行建模。目前，该领域的研究大多采用数学方法，如自动机[5]、状态图[6]、Petri 网[7,8]、图和接口自动机[9]，这些方法所使用的数学或理论工具大都具有复杂的符号表示和公式推理，对使用者的专业技能要求较高，其实用性大打折扣。

由于人们通常通过调用 Web 服务的操作来使用 Web 服务提供的功能，Web 服务合成是通过一组已经发布的 Web 服务的操作实现特定的功能。因此，研究 Web 服务合成方法实质上就是研究如何找到一组需要调用的 Web 服务的操作，以及如何以串行、并行等方式组合调用这些操作来完成期望的功能。这个功能可以用合成后得到的新的 Web 服务的一个操作来描述。

从这个意义上说，研究 Web 服务合成问题可以转化成研究如何将一组操作合成为一个大的操作的问题。操作可以由它的输入和输出来表示，一组操作可以抽象为一个表示输入输出依赖关系的依赖图，操作的输入和输出是图中的顶点，图中的边表示操作的调用，因此可以借助于图论中的方法研究 Web 服务合成。

定义 1　Web 服务的一个操作（标记为 p）定义为二元组 (I,O)，其中 $p.I$ 表示操作 p 的输入集合，$p.O$ 表示操作 p 的输出集合。

定义 2　一个 Web 服务 w 定义为三元组 (K,I,O)，$w.I=\bigcup_{p\in w.K} p.I$，$w.O=\bigcup_{p\in w.K} p.O$，$w.K=\{p \mid p$ 是 w 的操作$\}$。

定义 3　对于一个 Web 服务 w 及其任意一个操作 $p\in w.K$，二元组 (w,p) 表示对 w 的操作 p 的一次调用，Web 服务调用标记为 c。

定义 4　有向图 $G_w=(V,E)$ 表示集合 W 中 Web 服务的操作的输入输出依赖关系，其中 G_w 的顶点集合 $V=\bigcup_{w\in W} w.I \bigcup_{w\in W} w.O$，$G_w$ 的边集 E 满足以下性质：

(1) E 中的任意一条边 e 定义为四元组 (n,m,c,s)，其中元素 $n,m\in V$，n 表示 e 的开始顶点；m 表示 e 的目标顶点；元素 c 表示一个 Web 服务调用 (w,p)，s 表示合成的代价。

(2) 对于任意的顶点 $v_1,v_2\in V$，当且仅当在 W 中存在 Web 服务 w_1 并且存在 w_1 的操作 p_1，使得 $v_1\in p_1.I$，$v_2\in p_1.O$，则在 G_w 中连上一条 v_1 指向 v_2 的有向边 e_1，其中 $e_1.n=v_1$，$e_1.m=v_2$，$e_1.c=c_1$，$c_1=(w_1,p_1)$。

(3) 对于每一次操作，将有一个代价，定义为服务代价，并将它作为 G_w 中一条边的权，用四元组中的 s 表示。

图 2 举例说明 G_w 的构造方法：Web 服务 BI 有操作 getAuth 和 getBook，getAuth 的输入是 ISBN，输出为作者；getBook 输入为 ISBN，输出为书名。Web 服务 CoI 有操作 getC，它的输入是作者、书名，输出是国家。

定义 5　在图 G_w 中，设集合 $M\in 2^{G_w.V}$，若存在 $L\in 2^{G_w.E}$，使得 L 是 M 一个输入边集，则称顶点集合 $N=\bigcup_{c\in C_L} c.p.I$（由输入边集的存在，可知 N 中顶点都在 G_w 中）是顶点集合 M 的一个前驱顶点集合，记为

$Q(M)$。把 $Q(M)$ 中只能并行才能产生 M 的这些顶点叫 M 的并行前驱，记为 $h(M)$。

文中所提出的算法是根据权值来寻找最佳组合方案，这是一个通用的模型，只要根据不同的合成标准来定义边的权值，就会按照相应的标准来选择最佳操作进行合成。

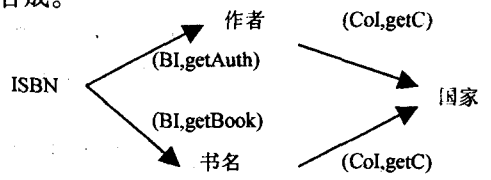

图 2 一个服务合成的依赖图

2 基于 Dijkstra 算法的最佳路径选择

2.1 Dijkstra 算法用于 Web 服务合成中的问题

Web 服务合成，实际上就是研究如何找到一组可以调用的 Web 服务操作，以及如何以串行、并行等方式组合调用这些操作来完成期望的功能。文中主要研究服务合成中的选择策略，目标是在已发现的服务的操作中找到某个组合序列产生一个新的服务，使最终合成的服务能满足用户的要求。这样可以将问题进一步简化，即知道输入、输出，求一条有向图中的路径，使这条路径的长度最短。如图 3 所示。

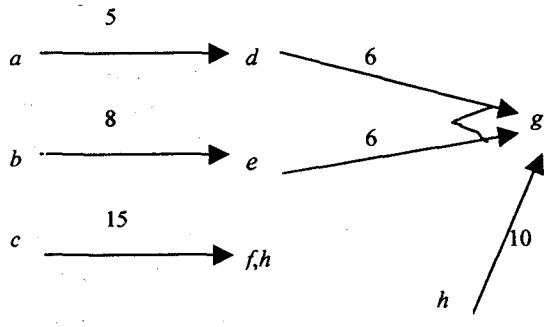

图 3 一个简化的服务合成有向图

在图 3 中，输入的是 a,b,c，要求得到 g。现在有两条路径可以得到：一条是 $c \to h \to g$，这条路径的代价是 25；另外一条路径是 $\{a,b\} \to \{d,e\} \to g$，也就是分别由 a,b 产生 d,e，后由 d,e 共同产生 g，图中的符号“<”表示共同产生，这条路径的总代价是 19。

通过以上的例子也可以看出：简化后的模型中实际上就是由求源点到目标点的一条长度最短的路径。Dijkstra 算法正是解决此类问题的首选算法，但是，Dijkstra 算法并不能直接用于 Web 服务合成。这是因为 Dijkstra 算法只能用于寻找点与点间的最短路径。而在 Web 服务的合成中可能出现要从多个点出发，去寻找最佳路径。这是因为，Web 服务合成一般有两种方式：一种是串行组合，一种是并行组合。所谓串行组合就是前一个操作的输出作为后一个操作的输入，这样一直到最终产生目标服务。并行组合就是目标服务同时由二个以上的串行组合所产生。比如，图 2 中 $c \to h \to g$ 就是一个串行组合，$\{a,b\} \to \{d,e\} \to g$ 就是一个并行组合。Dijkstra 算法只能用来确定串行合成的最短路径，而对并行合成则毫无办法。这是 Dijkstra 算法用于 Web 服务合成中的一个问题。其次，Web 服务合成中一个输入可能会产生出多个输出。如 c 可以产生出 f,h 两个节点，而刚好 h 可以产生目标 g，即在图中 f,h 节点和 h 节点实际上应该当作一个节点来考虑，Dijkstra 算法却没有这样的能力。

以上是 Dijkstra 算法应用于 Web 服务合成选择策略中遇到的问题，为解决这些问题，提出了一些办法。下面来讨论细节。

2.2 解决的方法

本算法是寻找一个最短的路径，根据上面的讨论，Web 服务的合成中，分串行合成和并行合成，串行合成的路径可以直接用 Dijkstra 算法来确定，而并行合成就不能直接用 Dijkstra 算法来处理，因为并行合成实际上就是多个输入同时产生一个输出，如上面的 d，e 同时产生 g，这样只需要保存这个输出的前驱，分别计算源点到这些前驱节点的最短路径。然后将这些最短路径与前驱到这个输出的代价相加即可得到并行组合的最短路径。这样就将求并行组合的最短路径转化成了求串行组合的最短路径问题，可以直接用 Dijkstra 算法计算。对于上面的第二个问题，只要在处理前将图中的多输出合并为一个新的节点处理，如果多输出中的某个输出是另外一个操作的输入，则可用新节点代替。

以上便是解决上节所遇问题的思路，下面给出数学表达式。$\delta(s,v)$ 表示 Web 服务合成中的最佳路径：

$$\delta(s,v) = \begin{cases} \min(\text{path}: s \xrightarrow{\text{path}} v) & s \text{ 到 } v \text{ 至少有一条路径} \\ \infty & \text{否则} \end{cases}$$

设节点 v 有前驱节点 m,n,u，其中 m,n 共同产生 v，而 u 可以独自产生 v，设 $\varepsilon(\{m,n\},v)$、$\varepsilon(u,v)$ 分别表示 m,n 共同产生 v 的代价和 u 独立产生 v 的代价，可得并行合成的路径长度为：

$$\delta_{并}(s,v) = \delta(s,m) + \delta(s,n) + \varepsilon(\{m,n\},v) \quad (1)$$

$$\delta_{串}(s,v) = \delta(s,u) + \delta(u,v)$$

$$\delta(s,v) = \begin{cases} \min(\delta_{并}(s,v), \delta_{串}(s,v)) & s \text{ 到 } v \text{ 至少有一条路径} \\ \infty & \text{否则} \end{cases} \quad (2)$$

对于图 2，设 $s = \{a,b,c\}, v = \{g\}$ 可得：

$\delta(s,v) =$

$\min(\delta(\{a,b,c\},d) + \delta(\{a,b,c\},e) + \varepsilon(\{d,e\},g)),\delta(\{a,b,c\},f) + \varepsilon(f,g))$

2.3 算法描述

最后给出了基于 Dijkstra 算法的 Web 服务合成最佳选择算法：

1)数据初始化并根据规则构造有向图，主要是初始化输入参数，也就是为初始化源节点集 s。搜索出图中有并行前驱的顶点，并保存其并行前驱。同时对多输出端进行合并，为下一步求解做准备。

2)用 Dijkstra 算法求源顶点集中的顶点到其他顶点的最小路径，$D[v]$ 存储所求源顶点集到顶点 v 的路径长度，$P[v]$ 存储源顶点集到顶点 v 的最小路径。$D[v] = \infty$ 表示源顶点集不可到达顶点 v 或者到顶点 v 的最短路径暂时未求出。

3) 对图中所有有并行前驱的顶点进行扫描，如果源点集到此顶点的并行合成的路径还未求出，由公式：$\delta_{并}(s,v) = \delta(s,m) + \delta(s,n) + \varepsilon(\{m,n\},v)$ 可得：$D'[v] = D[m] + D[n] + \varepsilon(\{m,n\},v)$，$D'[v]$ 表示其并行合成的路径，然后根据公式(2) 将 $D'[v]$ 与 $D[v]$(前次循环所求的最小路径，即串行合成的最小路径) 比较，用两者中较小的值更新 $D[v]$，同时更新 $p[v]$。

4) 判断上次执行中 D 和 P 中的值是否被改变，若否转入 5)，是则转入 2)。

5) 根据期望输出在 $D[v]$ 和 $P[v]$ 中查找所需要合成最佳路径及路径长度。

6) 输出结果，结束。

2.4 算法效率分析

Dijkstra 算法的时间复杂度为 $O(n^2)$，本算法是循环执行 Dijkstra 算法，当情况最坏时，Dijkstra 算法将被调用 n 次，因此本算法的时间复杂度为 $O(n^3)$。

3 实验结果及分析

为了验证算法的可行性和效率，用 Java 语言及 prolog 库实现了上述算法并集成到一个现成的 Web 服务合成系统中，进行了相关的测试。测试平台为：主频 3.06GHz 的 PC 机(512M 内存)，Windows Server 2000 操作系统。实验考察了算法在两种情况下的性能，首先固定用户输入的个数，即固定源点集中的顶点个数，改变可以用于合成的规则数量，也就是改变有向图中顶点的个数和有向图中边的数量，然后固定规则数目，而改变用户输入参数。图 4 给出了算法执行的时间曲线图。从中可以看出，基于 Dijkstra 算法的选择策略中所用算法随着有规则数目的增长而呈指数增长。图 5 给出了固定规则数目，而改变用户输入参数，基于 Dijkstra 算法的选择策略中所用算法的时间变化幅度并不大。在近二百多次实验中，该算法都能准确地选择出最佳操作，产生最优的服务合成方案。

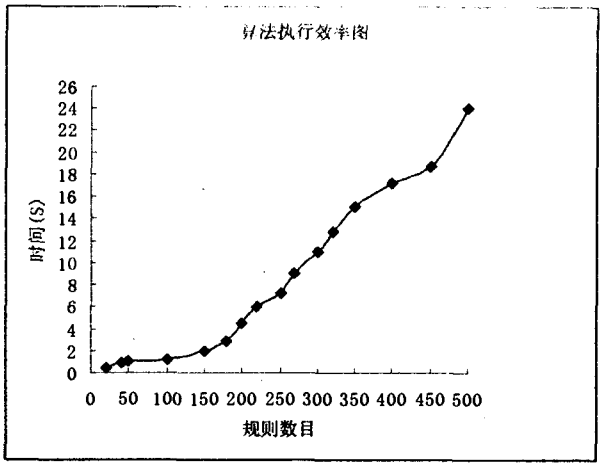

图 4　算法执行效率曲线图一

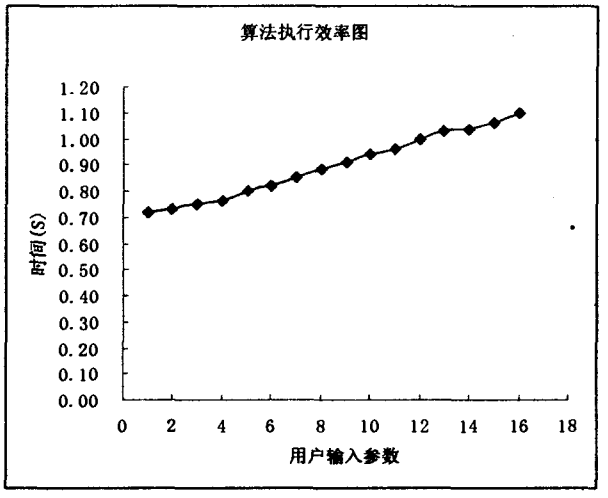

图 5　算法执行效率曲线图二

实验表明，基于 Dijkstra 算法的最佳选择策略在虽然在效率上和目前主流合成系统的所用选择策略还存在一定的差距，但准确性有所提高，能准确地选择最合适的操作进行服务合成。

4 结束语

文中根据 Web 服务合成的特点，应用图论中的相关知识建立了服务合成的模型表示。将 Web 服务合成转化为求有向图中起始顶点到目标顶点的最短路径问题，然后基于 Dijkstra 算法提出了一种选择策略算法，此算法能够按照相应的选择标准选择出最恰当的操作进行服务的合成，从而产生用户需要的复合服务。实验表明，该算法是可行的，下一步将进一步改进算法

(下转第 111 页)

不需要进行整个页面的渲染与刷新,而仅仅需要执行这段脚本内容,将页面的控件进行更新即可[4]。

而且所有的标准组件都支持 Ajax,对于不支持 Ajax 的第三方组件,可以使用<ajax:renderGroup>的标签转换。Apusic JSF 引擎实现了一个 Ajax Render Kit,在 HTML 文档中嵌入 Java Script 代码来实现 Ajax 特性呈现,而替换 Render Kit 只需要修改配置文件[5]。

Apusic JSF 实现了一个<ajax:status>的标签,用于接受发送和完成 Ajax 请求时触发的事件。缺省的实现是在发送 Ajax 请求之前显示一个 HTML 片段,在完成 Ajax 请求之后显示另一个 HTML 片段。开发人员可以设置<ajax:status>标签的 onStart 和 onStop 属性,来自定义修改执行的 Java Script 代码以实现更复杂的效果。此外,还实现了一个<ajax:invoke>标签,能采用 RPC(Remote Procedure Call)方式调用服务器端 Java 对象的方法[5]。

5　其它特性

Apusic JSF 还包含其它特性,如控件的换肤功能,控件对 IE、Mozilla(Firefox)、Opera 等多浏览器的支持,以及强大的布局功能等[5]。

6　结束语

JSF 事实上与 MVC 模式有所不同,它是 Web 开发。编程人员的开发理念更为轻松。该模型使应用程序开发人员能够设计应用程序的页面流。与 Struts 的方式类似,所有的页面流信息都定义在 JSF 配置 XML 文件(faces - config. xml)中,而非硬编码在应用程序中。这很大程度简化了开发人员开发难度,简化了应用程序的开发。

JSF 自定义组件由 java 代码和 tag 库文件组成,开发难度应该与现有 I2SS 组件开发的难度基本一致。自定义组件通过自定义标记构造页面,在页面上增加组件的数量会对性能有较大的影响。在定制组件功能上,Apusic Studio 可以考虑增加设计导航功能,并使应用配置文件与组件功能和呈现设计保持同步。

JSF 是可高度抽象、可扩展的 Web 表现层框架级解决方案。但是受客户端浏览器技术的限制,并未达到无缝、快捷的用户体验。JSF 的组件和事件模型与 Ajax 的异步通讯方式相配合,优势互补,会使互联网客户端应用环境界面更加友好便捷。

参考文献:

[1] Crane D. Ajax 实战[M]. 北京:人民邮电出版社,2006.

[2] Harrop R, Machacek J. Pro Spring 中文版[M]. 夏　昕译. 北京:电子工业出版社,2006.

[3] Geary D. A first look at JavaServer Faces, Part 1[EB/OL]. 2002 - 11 - 29. http://www.javaworld.com/javaworld/jw - 11 - 2002/jw - 1129 - jsf.html? page = 1.

[4] Kurniawan B. JavaServer Faces Programming[M]. 北京:清华大学出版社,2005.

[5] 袁红岗. JavaEE without Ajax[EB/OL]. 2007 - 05 - 28. http://www.apusic.com/article/article19.htm.

(上接第 107 页)

以达到效率和准确性的平衡。

参考文献:

[1] Casati F, Llnicki S, Jin L, et al. Adaptive and Dynamic Service Composition in eFlow[C]//Proceedings of the 12th International Conference on Advanced Information Systems Engineering, 2000. London: Springer Verlag, 2000: 13 - 31.

[2] Zeng L, Benatallah B, Nguyen P, et al. AgFlow: Agent - based Cross Enterprise Workflow Management System[C]//Proceedings of the 27th Intl. Conf. on Very Large Data Bases, 2001. Italy: Morgan Kaufmann Publishers Inc, 2001: 697 - 698.

[3] Benatallah B, Dumas M, Sheng Q. The Self - Serv Environment for Web Services Composition[J]. IEEE Internet Computing, 2003, 7(1): 40 - 48.

[4] 蒋运承,汤　庸. 服务组合的质量估计模型[J]. 小型微型计算机系统,2006,27(8):1519 - 1524.

[5] Berardi D, Calvanese D, Giacomo G, et al. Automatic Composition of E - Services That Export Their Behavior[C]//Proceedings of the 1st Intl Conf on Servi ce - Oriented Computing, 2003. Berlin: Springer Verlag, 2003: 43 - 58.

[6] Benatallah B, Dumas M. Declarative Composition and Peer - to - Peer Provisioning of Dynamic Web Services[C]//Proceedings of the 18th Intl. Conf. Data on Engineering, 2002. Washington: IEEE Computer Society, 2002.

[7] Hamadi R, Benatallah B. A Petri net - based model for web service composition[C]//Proceedings of the 14th Australasian Database Conf, 2003. Darlinghurst: Australian Computer Society, 2003: 191 - 200.

[8] Narayanan S, Mcllraith S A. Simulation, Verfication and Automated Composition of Web Services[C]//Proceedings of the 11th intl Conf on World Wide Web, 2002. New York: ACM Press, 2002: 77 - 88.

[9] Hashemaian S V, Mavaddat F. A Graph - Based Approach to Web Services Composition[C]//Proceedings of the 2005 Symposium on Applications and the Internet, 2005. Washington: IEEE Computer Society, 2005: 183 - 189.