一个基于有色 Petri 网的自动 Web 服务合成模型*)

张昭理　洪　帆
(华中科技大学计算机科学与技术学院　武汉 430074)

摘　要　提出了一个基于有色 Petri 网的自动 Web 服务合成模型，为 Web 服务的合成提供语义支持，提高合成服务的可靠性和可维护性。该模型将服务的合成结构分成顺序、并发、选择、循环、置换 5 种合成结构。给出了 Web 服务基于有色 Petri 网的形式化定义。定义了一个封闭的 Web 服务合成算法，通过算法获得的框架能够对 Web 服务进行说明性的合成。定义了一个自动 Web 服务合成算法，通过该算法，可以对 Web 服务进行有人工指导的半自动合成和无人工指导的自动合成。

关键词　Web 服务，有色 Petri 网，自动 Web 服务合成，模型

A Colored Petri Net-based Model for Automated Web Service Composition

ZHANG Zhao-li　HONG Fan
(School of Computer, Huazhong University of Science and Technology, Wuhan 430074, China)

Abstract　In this Paper, we propose a colored Petri net-based model for Web service composition to provide the semantic support for automated Web service composition, and improve the reliability and maintainability of composite services. The composite constructs in the model are sequence, concurrent, choice, loop and replace. The Web service is formally defined by colored Petri net. A closed composing algebra is defined to obtain a framework which enables declarative composition of Web services. An automated composing algebra is defined to enable automated composition and semi-auto composition with manual operations.

Keywords　Web service, Colored petri net, Automated Web service composition, Model

1　引言

Web 服务这个概念最近变得非常流行，但是至今仍没有一个统一的清晰定义。本文认为 Web 服务是一个由 Web 服务描述语言(WSDL，Web Service Description Language)描述的软构件，并且可以由标准网络协议通过 HTTP 访问，例如 SOAP 协议(Simple Object Access Protocol)，但不限于这个协议。Web 服务应该是基于公开标准，独立于平台和具体应用的，并且能够分享数据与资源。Web 服务合成是将现有的 Web 服务连接与合并起来，生成新的 Web 服务，为现有的服务集合增加价值。当前基于 UDDI(Universal Description, Discovery, and Integration)，WSDL 和 SOAP 的技术无法实现复杂 Web 服务的合成，只能为 Web 服务合成提供有限的支持[1-3]。

近年来，Web 服务的数量在惊人地增长，并且在不断地更新。Web 服务合成是一个非常复杂的任务，已经不可能由人工来完成。为 Web 服务提供自动或半自动的合成模型和工具变得非常重要[4]。因此，本文提出了一个基于有色 Petri 网 CPN[5]的算法为 Web 服务建模。这个模型有足够的表达能力，能够表达复杂 Web 服务合成的语义以及各自的定义。通过算法获得的框架能够对 Web 服务进行说明性的自动合成。

2　基于 CPN 的 Web 服务合成

为了快速计算，很多研究者[6-8]倾向于使用 Petri 网。因为 Petri 网非常适合描述 Web 服务中的流关系，为 Web 服务的分布式特性建模，以及表达 Web 服务中的方法来推导流关系的正确性。

基于库所/变迁网的 Web 服务合成模型[7,10]不能体现输入、输出数据的类型和个性，因此无法分析合成服务的有效性，也无法提供 Web 服务的自动合成框架。近期的研究中，有研究者提出了 Web 服务组合的有色网模型[9]，但该模型也没有为 Web 服务提供自动的合成工具。在面向消息和基于行为的 Petri 网模型[10]中，Web 服务的合成需要用户提出明确的需求，定义严格的行为特性，实际上是人工指导下的服务合成。

2.1　作为有色 Petri 网的 Web 服务

一个 Web 服务的行为基本上是一个偏序的操作集。因此，它可以直接被映射为一个 Petri 网。操作被映射为变迁，Web 服务的状态被映射为库所。库所与变迁之间的箭头用来表示通常的关系。

令描述 Web 服务的 Petri 网包括一个输入库所(一个没有输入弧的库所)和一个输出库所(一个没有输出弧的库所)[7]。一个 Petri 网的输入库所用来吸收信息，而输出库所则用来释放信息。输入库所和输出库所的引入将有助于定义 Web 服务合成运算符以及对某些性质(例如可达性，死锁和活性)的分析。在任何时候，一个 Web 服务将处于下列状态之一：未实例化(NotInstantiated)，就绪(Ready)，执行(Running)，阻塞(Suspended)或完成(Completed)。当一个 Web 服务处在就绪状态，就意味着在相应输入库所中的托肯使得输

*)国家自然科学基金(60403027)资助项目。张昭理　博士研究生；洪　帆　教授，博士生导师。

入库所的后集(变迁集)可以发生。而完成状态则意味着输出库所的前集已经发生,并且在相应的输出库所中产生了托肯。

定义 1 (服务网,Service Net)

$SN = (P, T, F, C, I_-, I_+, M_0, i, o, l)$ 称为服务网的充要条件是:

$\Sigma = (P, T, F, C, I_-, I_+, M_0)$ 是一个有色 Petri 网;

$i \in P$ 是输入库所,且 $\cdot x = \emptyset$;

$o \in P$ 是输出库所,且 $x \cdot = \emptyset$

$l: T \to A \cup \{\tau\}$ 是一个标签函数,其中 A 是一个操作名称的集合。令 $\tau \notin A$ 代表静操作。

$C: P \cup T \to Pow(D)$,$Pow(D)$ 为颜色集 D 之幂集合,使得:对于 $p \in P$,$C(p)$ 是库所 p 上所有可能的托肯色之集合;对于 $t \in T$,$C(t)$ 是变迁 t 上所有可能的出现色之集合。

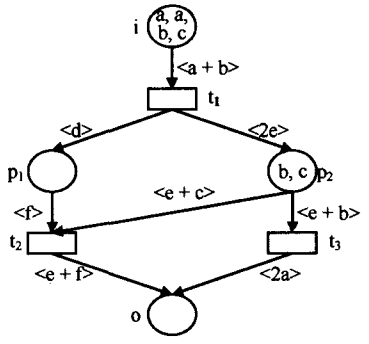


图 1 服务网示例

静操作是变迁,但它的发生不能被观察到。它们被用来区分服务的内部行为和外部行为。服务网的示例显示在图 1 中。变迁的输入弧上标注的是令这个变迁发生所需要托肯的数量和颜色,变迁的输出弧上标注的是这个变迁发生后产生托肯的数量和颜色。托肯的颜色表示信息的类型,而并不是表示信息的内容。因此具有相同颜色的两个托肯可能是不同的。显然图中的 t_1 可以发生,这个服务网处于就绪状态。然后,当这个服务网到达完成状态后,输出库所 o 中应该有 4 个托肯:两个颜色为'a',一个颜色为'e',一个颜色为'f'。下面将给出 Web 服务的定义。

定义 2 (Web 服务)

一个 Web 服务是一个多元组 $S = (ID, Desc, Loc, URL, CS, SN)$,其中:

ID 是这个服务的唯一标识,它可以是名字或全局唯一编号;

$Desc$ 是对所提供服务的描述,它概括描述这个服务所提供的东西;

Loc 是这个服务所处的服务器;

URL 是这个服务的调用地址;

CS 是构成这个服务的服务组件所组成的集合,如果 $CS = \{ID\}$,那么 S 是一个基本(原子)服务,否则 S 是一个合成服务;

$SN = (P, T, F, C, I_-, I_+, M_0, i, o, l)$ 是服务网,对这个服务的动态行为建模。

2.2 合成算法与结构

一个 Web 服务要完成特定的任务,很可能依赖于其他的 Web 服务,因此有必要将它们合成起来。合成两个或多个服务而产生的一个新服务,不仅可以提供原来单个服务的行为逻辑,而且可以完成新的任务。这就意味着,现有的服务可以相互协作,尽管这种协作并非事先就设计好的。服务合成可以是静态的(服务组件以一种事先协商好的方式进行互动),也可以是动态的(它们互相发现对方,然后开始协商)。

在此节中,将介绍合成算法,利用这个算法可以将现有的服务作为集成块创建增值的新服务。相关的研究者已经讨论过多种合成结构[7-13]。本文将顺序结构(sequence),并发结构(concurrent),选择结构(choice),和循环结构(loop)视为控制流中的基本结构;将置换结构(replace)视为高级结构。

合成结构将用来进行基本和高级的 Web 服务合成。服务的集合定义如下:

$$S ::= \varepsilon \mid X \mid Seq(S, S) \mid Conc(S, S) \mid Choice(S, S) \mid Loop(S) \mid Rep(S, a, S) \mid$$

令 $S_i = (ID_i, Desc_i, Loc_i, URL_i, CS_i, SN_i)$,其中 $SN_i = (P_i, T_i, F_i, C_i, I_{-i}, I_{+i}, M_{0i}, i_i, o_i, l_i)$,$i = 1, \cdots, n$ 是 n 个 Web 服务,并且对 $i \neq j$ 有 $P_i \cap P_j = \emptyset$ 和 $T_i \cap T_j = \emptyset$。

需要注意的一点是,将在下面描述的服务合成只适用于句法上不同的服务。这是因为,各个服务要正确地合成,它们之间必须是不相交的。但是,一个服务有可能与它自己合成。在这种情况下,必须在合成之前解决重叠的问题。这个问题可以通过对这两个相同服务的其中之一重命名集合 P 和 T 来解决。这两个服务在库所和变迁的名称上仍然保持同构。另外,静操作在图形中用黑色的方块来表示相应的变迁。

空服务 空服务 ε 是一个不执行任何操作的服务。使用它是出于技术和理论研究上的需要。

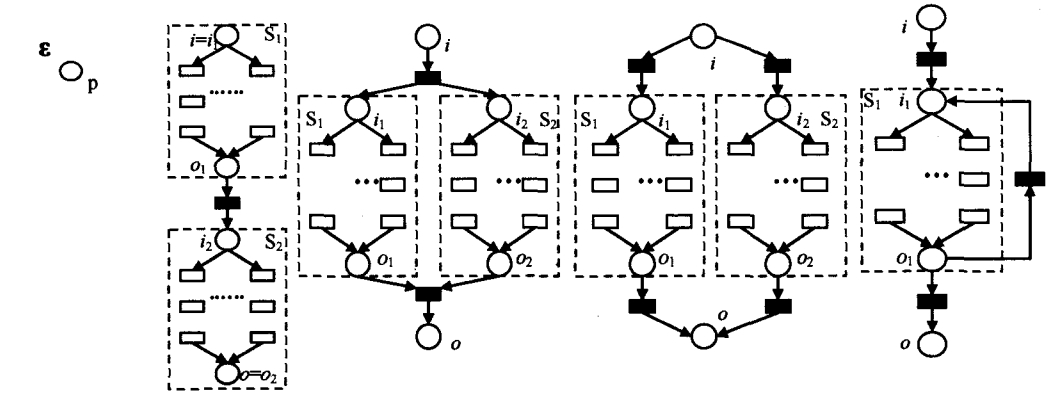


图 2 基本合成结构的有色 Petri 网

定义 3-1 空服务定义为 $\varepsilon = (ID, Desc, Loc, URL, CS, SN)$,其中:$ID$ = Empty;$Desc$ = 空服务;Loc = Null,意味着该服务没有服务器;URL = Null,意味着该服务没有 URL;$CS = \{\text{Empty}\}$;$SN = (\{p\}, \varnothing, \varnothing, C, 0, 0, \{0\}, p, p, \varnothing)$。

ε 的有色 Petri 网图形表示显示在图 2(a)中,只包含一个库所。

除了空服务以外,在以下定义中,ID 表示合成服务的唯一标识;$Desc$ 表示合成服务的描述;Loc 表示合成服务所处的服务器;URL 表示合成服务的调用地址。

顺序结构 顺序操作符使得服务 S_1 和服务 S_2 顺序执行。S_1 完成之后 S_2 才能开始执行。顺序结构典型的情况是一个服务依赖另一个服务的输出。

定义 3-2 服务 $Seq(S_1, S_2)$ 定义为 $Seq(S_1, S_2) = (ID, Desc, Loc, URL, CS, SN)$,其中:

$CS = CS_1 \cup CS_2$,

$SN = (P, T, F, C, I_-, I_+, M_0, i, o, l)$,其中:

$P = P_1 \cup P_2$, $T = T_1 \cup T_2 \cup \{t\}$, $F = F_1 \cup F_2 \cup \{(o_1, t), (t, i_2)\}$, $i = i_1$, $o = o_2$, $I_- = I_- \cup I_-(o_1, t)$, $I_+ = I_+ \cup I_+(i_2, t)$, $M_0 = M_{01} \cup M_{02}$, $l = l_1 \cup l_2 \cup \{(t, \tau)\}$

给定 S_1 和 S_2,$Seq(S_1, S_2)$的有色 Petri 网图形表示显示在图 2(b)中。

并发结构 并发操作符使得两个服务 S_1 和 S_2 并发执行。并发结构特别典型的情况是,一些互不干扰的小(原子)服务可以合并到一起形成一个大的合成服务。

定义 3-3 服务 $Conc(S_1, S_2)$定义为 $Conc(S_1, S_2) = (ID, Desc, Loc, URL, CS, SN)$,其中:

$CS = CS_1 \cup CS_2$,

$SN = (P, T, F, C, I_-, I_+, M_0, i, o, l)$,其中:

$P = P_1 \cup P_2 \cup \{i, o\}$, $T = T_1 \cup T_2 \cup \{t_i, t_o\}$, $F = F_1 \cup F_2 \cup \{(i, t_i), (t_i, i_1), (t_i, i_2), (o_1, t_o), (o_2, t_o), (t_o, o)\}$, $I_- = I_- \cup I_-(i, t_i) \cup I_-(o_1, t_o) \cup I_-(o_2, t_o)$, $I_+ = I_+ \cup I_+(i_1, t_i) \cup I_+(i_2, t_i) \cup I_+(o, t_o)$, $M_0 = M_{01} \cup M_{02}$, $l = l_1 \cup l_2 \cup \{(t_i, \tau), (t_o, \tau)\}$

给定 S_1 和 S_2,$Conc(S_1, S_2)$的有色 Petri 网图形表示显示在图 2(c)中。

选择结构 选择操作符使得服务 S_1 或 S_2 执行。一旦其中一个被执行,则另一个就无效了。

定义 3-4 服务 $Choice(S_1, S_2)$定义为 $Choice(S_1, S_2) = (ID, Desc, Loc, URL, CS, SN)$,其中:

$CS = CS_1 \cup CS_2$,

$SN = (P, T, F, C, I_-, I_+, M_0, i, o, l)$,其中:

$P = P_1 \cup P_2 \cup \{i, o\}$, $T = T_1 \cup T_2 \cup \{t_{i1}, t_{i2}, t_{o1}, t_{o2}\}$, $F = F_1 \cup F_2 \cup \{(i, t_{i1}), (i, t_{i2}), (t_{i1}, i_1), (t_{i2}, i_2), (o_1, t_{o1}), (o_2, t_{o2}), (t_{o1}, o), (t_{o2}, o)\}$, $I_- = I_- \cup I_-(i, t_{i1}) \cup I_-(i, t_{i2}) \cup I_-(o_1, t_{o1}) \cup I_-(o_2, t_{o2})$, $I_+ = I_+ \cup I_+(i_1, t_{i1}) \cup I_+(i_2, t_{i2}) \cup I_+(o, t_{o1}) \cup I_+(o, t_{o2})$, $M_0 = M_{01} \cup M_{02}$, $l = l_1 \cup l_2 \cup \{(t_{i1}, \tau), (t_{i2}, \tau), (t_{o1}, \tau), (t_{o2}, \tau)\}$

给定 S_1 和 S_2,$Choice(S_1, S_2)$的有色 Petri 网图形表示显示在图 2(d)中。

循环结构 循环操作符使得服务 S 执行一定的次数。循环结构特别典型的情况是在通信或者质量控制的时候,服务需要执行多次。

定义 3-5 服务 $Loop(S_1)$定义为 $Loop(S_1) = (ID, Desc, Loc, URL, CS, SN)$,其中:

$CS = CS_1$,

$SN = (P, T, F, C, I_-, I_+, M_0, i, o, l)$,其中:

$P = P_1 \cup \{i, o\}$, $T = T_1 \cup \{t_i, t_o, t\}$, $F = F_1 \cup \{(i, t_i), (t_i, i_1), (o_1, t_o), (t_o, o), (o_1, t), (t, i_1)\}$, $I_- = I_- \cup I_-(i, t_i) \cup I_-(o_1, t_o) \cup I_-(o_1, t)$, $I_+ = I_+ \cup I_+(i_1, t_i) \cup I_+(o, t_o) \cup I_+(i_1, t)$, $M_0 = M_{01}$, $l = l_1 \cup \{(t_i, \tau), (t_o, \tau), (t, \tau)\}$

给定 S_1,$Loop(S_1)$的有色 Petri 网图形表示显示在图 2(e)中。

置换结构 在置换结构中,一些操作被更细节化的非空服务替换。这可以用来在服务中引入新的服务组件。置换是将设计从一个高级的抽象形式转换到一个低级的更具体的形式,从而能够进行层次化的建模。

定义 3-6 令 $a \in A$,服务 $Rep(S_1, a, S_2)$定义为 $Rep(S_1, a, S_2) = (ID, Desc, Loc, URL, CS, SN)$,其中:

如果 $a \in l_1(T_1)$则 $CS = CS_1 \cup CS_2$,否则 $CS = CS_1$;

$SN = (P, T, F, C, I_-, I_+, M_0, i, o, l)$,其中:

$P = P_1 \cup (P_2 - \{i_2, o_2\})$, $T = T_1 \cup T_2 - l_1^{-1}(a)$, $F = (F_1 - \{(x, y) \mid x \in l_1^{-1}(a) \text{ or } y \in l_1^{-1}(a)\}) \cup F_2$, $I_- = (I_{-1} - \{I_{-1}(p, t) \mid t \in l_1^{-1}(a)\}) \cup I_{-2}$, $I_+ = (I_{+1} - \{I_{+1}(p, t) \mid t \in l_1^{-1}(a)\}) \cup I_{+2}$, $M_0 = M_{01} \cup M_{02} - M(i_2) - M(o_2)$, 如果 $t \in (T_1 - l_1^{-1}(a))$,则 $l(t) = l_1(t)$;否则 $l(t) = l_2(t)$

给定 S_1 和 S_2,$Rep(S_1, a, S_2)$ 的有色 Petri 网图形表示显示在图 3 中。从定义以及图形表示中可以发现,被标定为将要被置换的变迁(操作)应该只能有一个输入弧和一个输出弧。在一个 Petri 网中如果每个变迁都只有一个输入弧和一个输出弧,这个 Petri 网就称为平凡 Petri 网(ordinary Petri net)。我们推荐高层次的设计形成平凡 Petri 网,这样就可以进行置换和层次化的建模。

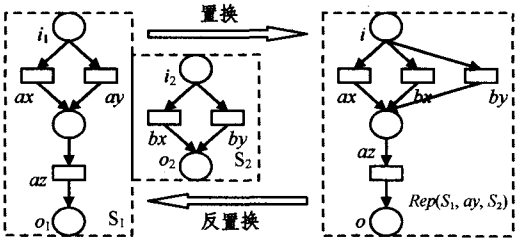


图 3 转换结构的有色 Petri 网

可以证明上述算法具有封闭性。它保证了对服务的每一个运算的结果都是一个服务,可以继续应用这些运算符。这样,就能够通过上述算法集成和重用现有的服务来建造更复杂的服务。

3 自动 Web 服务合成

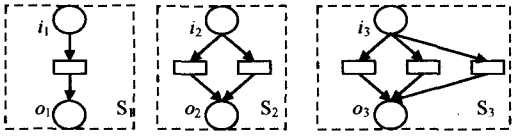
自动 Web 服务合成的复杂性主要来自于以下几个方面:首先,Web 服务数量的增长非常迅速,Web 服务库变得越来越庞大。其次,Web 服务的内容更新也非常快,而服务合成需要利用各个服务最新的信息。这使得要根据不同用户的需要利用人工完成 Web 服务的合成几乎不可能。最后,Web 服务由不同的组织和公司开发出来,使用的是不同的模型和语言[4]。因此,需要一个高层次的抽象模型和算法来解决自动 Web 服务的合成问题。

3.1 Web服务合成的有效性

要实现自动Web服务合成，必须首先判断哪些服务以何种结构可以合成到一起。即Web服务合成的有效性问题。

在一个CPN模型中，颜色（数据类型）是可区分的，因此设计者可以根据输入、输出库所的颜色来判断合成服务的有效性。与面向对象程序中一样，托肯的颜色被定义为可以继承的类。在以下叙述中，“颜色”和“数据类型”将代表相同的含义。例如，‘c’和‘d’分别继承自基类‘a’和‘b’，则需要将其表示为‘a-c’和‘b-d’以体现继承关系。因此，当一个服务需要颜色为‘a’的输入，那么‘a’和‘a-c’都将被接受；然而，当一个服务需要颜色为‘a-c’的输入，则只有‘a-c’被接受，‘a’将被拒绝。

考虑如图4所示的三个服务。可以发现S_1不能够通过循环结构与其自身合成，因为它的输出库所无法为其自身的输入库所提供颜色为‘a’的托肯。同样原因，S_2也不能通过循环结构与自身合成，而S_3可以。另外，从图4还可以发现，只有S_1在前，S_2在后的情况下，S_1和S_2才能通过顺序结构合成。而S_2和S_3可以以任意顺序通过顺序结构合成在一起。



$C(i_1)=\{a\}, C(i_2)=\{b\text{-}f, c\}, C(i_3)=\{b, c\}$
$C(o_1)=\{b\text{-}d, c\}, C(o_2)=\{b\text{-}e, a, c\}, C(o_3)=\{b\text{-}f, c\}$

图4 Web服务合成的有效性

通过以上分析，可以得到以下规则：

（1）如果合成服务$S = Seq(S_1, S_2)$是有效的，则必有$C(i_2)\subseteq C(o_1)$；

（2）如果合成服务$S = Conc(S_1, S_2)$是有效的，则必有$C(i_1)\cup C(i_2)\subseteq C(i)$且$C(o)=C(o_1)\cup C(o_2)$；

（3）如果合成服务$S = Choice(S_1, S_2)$是有效的，则必有$C(i_1)\cup C(i_2)\subseteq C(i)$且$C(o)=C(o_1)\cup C(o_2)$；

（4）如果合成服务$S = Loop(S_1)$是有效的，则必有$C(i_1)\subseteq C(o_1)$。

基于库所/变迁网的Web服务合成模型[7, 10]不能体现输入、输出数据的类型和个性，因此无法分析合成服务的有效性，因此也无法提供Web服务的自动合成框架。

3.2 半自动服务合成

普遍情况下，Web服务的用户只知道他需要通过服务得到什么结果，而并不知道服务库中有哪些服务，以及这些服务是如何组织的。因此需要根据用户要求为用户提供候选的服务，由用户自己选择合适的服务来合成以得到期望的结果。

考虑如图5所示的一个实例。一个用户希望知道美国城市休斯顿的气温，根据用户的要求，系统会提供输出数据类型为气温(Fahrenheit)的一系列服务供用户选择。当用户选择了服务$S_1(ID$ = Get Temperature)之后发现，该服务需要使用邮政编码(数据类型Zipcode)作为输入。但是用户并不知道休斯顿的邮政编码，因此系统会提供输出数据类型为Zipcode的一系列服务供用户选择。当用户选择了服务$S_2(ID$ = Find Zipcode)之后发现，该服务需要输入英文城市名Houston。但是用户并不知道如何拼写Houston，因此系统又会提供一系列服务供用户选择。最后用户选择了字典服务$S_3(ID$ = Dictionary)，并输入中文“休斯顿”，经过三个Web服务的合作最终得到了所期望的结果。

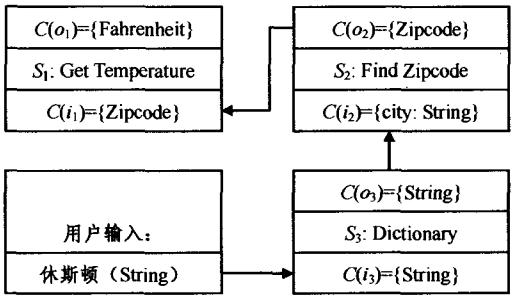


图5 半自动服务合成

可以看出，这三个Web服务要通过顺序结构进行合成，以完成用户的要求。根据上一小节的合成服务有效性，这三个服务必须满足$C(i_2)\subseteq C(o_3)$和$C(i_1)\subseteq C(o_2)$。因此，为用户提供候选服务比较简单，只需要在服务库中选取那些输出库所的数据类型与用户要求的数据类型相匹配的服务即可。设用户要求的输出为*output*，则第一个候选服务S_1需要满足$C(output)\subseteq C(o_1)$；若$S_1$不能完成用户的任务，则下一个候选服务$S_2$需要满足$C(i_1)\subseteq C(o_2)$；以此类推。服务系统除了为用户提供候选服务外，还需要提供对Web服务的描述Desc以帮助用户进行选择。

3.3 自动服务合成

在另外一些情况下，Web服务是由应用程序自动调用，或者用户要求完成的任务提供了特定的输入和输出，这时就需要系统提供没有人工参与的自动服务合成。

设用户输入为*input*，输出为*output*，如果服务库中存在一组Web服务$S_1, S_2, \cdots, S_n$，满足：

$C(output)\subseteq C(o_1), C(i_1)\subseteq C(o_2), \cdots, C(i_{n-1})\subseteq C(o_n), C(i_n)\subseteq C(input)$

则这些Web服务就可以通过顺序结构合成，形成一条服务链，以完成用户的任务。其中的Web服务既可以是基本服务，也可以是合成服务。对于满足上述要求的合成服务定义如下：

定义4(有效合成服务)

设全部Web服务的集合为S，给定输入*input*和输出*output*，有效合成服务*ACS*是满足下列条件的一个合成Web服务：

（1）*ACS*由服务$S_1, S_2, \cdots, S_n$通过并发结构合成，对于$i = 1, \cdots, n$有$S_i\in S$；

（2）$C(output)\subseteq C(o_1)\cup C(o_2)\cdots\cup C(o_n)$；

（3）$\{S_1, S_2, \cdots, S_n\}$的任何真子集均不能满足条件(2)。

集合$\{S_1, S_2, \cdots, S_n\}$可以是多重集。因此，对于给定输出$C(output) = \{a, b, b\}$，组件服务集合可能是$\{S_1, S_2, S_2\}$，其中$S_1$的输出为$a$，而$S_2$的输出为$b$。在实际的操作中，可以用循环结构代替并发结构完成相同组件服务的合成。这并不影响有效合成服务的定义。

从给定的输出*output*出发，不断寻找匹配的有效合成服务，使得前一个服务的输入是后一个服务输出的子集，直到最后一个服务的输入是*input*的子集。则这些Web服务就可以通过顺序结构合成，形成一条服务链，以完成用户的任务。每次寻找的结果可能有多个匹配服务，因此整个寻找过程会形成一个服务合成树。

服务合成树是由其构造算法定义的。为了在定义中全部采用服务来描述，将 *input* 包装为一个输入服务 e，其中 $i_e = \emptyset, o_e = input$；将 *output* 包装为一个输出服务 r，其中 $i_r = output$，$o_r = \emptyset$。

定义 5(服务合成树)

对于给定的输入服务 e 和输出服务 r，服务集合 S 的服务合成树 $T(S)$ 是由下述算法构造的树结构，它的每一个节点都是一个服务。

算法步骤如下：

(1) $T(S)$ 的初值只有根节点 r；

(2) 令 x 为 $T(S)$ 的叶节点。若 $x = \emptyset$ 或者 $x = e$ 则 x 为真叶节点。若 $T(S)$ 的所有叶节点都是真叶节点，算法终止；否则执行(3)；

(3) 若 $T(S)$ 有叶节点 x，且 x 不是真叶节点，若 $C(i_x) \subseteq C(o_e)$，则为 x 添加一个子节点 $y = e$；否则，若存在符合定义 4 的有效合成服务 y，使得 $C(i_x) \subseteq C(o_y)$ 且 y 与从 r 到 x 的路径中的所有服务均不相同，则将 y 添加为 x 的子节点；否则，添加 $\emptyset$ 为 x 的子节点；

(4) 回到步骤(2)。

如果服务集合 S 中服务个数是有限的，则算法会自然终止。步骤(3)中要求 y 与从 r 到 x 的路径中的所有服务均不相同，是为了避免出现环。如图 6 所示，对于真叶节点 $x = e$，则存在从根节点 r 到 x 的一条路径，该路径上的服务(可以是基本服务或合成服务)将形成一条服务链，以完成用户的任务。若所有真叶节点都为 $\emptyset$，则这样的服务链是不存在的。

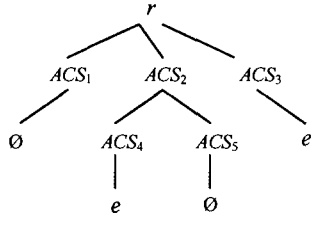


图 6　服务合成树

对于 $T(S)$ 上的每个合成服务 ACS_i，均是 S 的子集。有 n 个服务的集合 S 的子集有 2^n 个，因此在的每一层上，算法复杂度将达到 $O(2^n)$，若 $T(S)$ 有 m 层，最坏情况下总的复杂度将达到 $O(2^{nm})$。这个复杂度是相当高的。降低复杂度的第一个方法是参数的精确匹配。例如姓名(name)，地址(address)，城市(city)等都是字符串(string)类型。如果用 string 类型匹配，则结果会很多，但是如果用精确类型 name，address，city 匹配，则结果就会少很多。第二个方法是采用基于语义的互操作，利用 Ontology 来描述 Web 服务，使得服务之间能够理解互相交换的信息[14, 15]。语义 Web 服务不是本文讨论的范围。第三种方法是设定 $T(S)$ 的最大深度。无论是否找到有效的服务链，当达到最大深度时算法自动终止。一般来说，当服务合成树深度超过 5～10，则生成的合成服务过于复杂，而且计算的响应时间也会非常长。因此设定 $T(S)$ 的最大深度可以在存在大量候选服务时降低复杂度。

结束语　本文提出了一个基于有色 Petri 网的自动 Web 服务合成模型，为 Web 服务的合成提供语义支持，提高合成服务的可靠性和可维护性。该模型的 Web 服务合成结构包括顺序、并发、选择、循环、置换 5 种合成结构。给出了服务网和 Web 服务基于有色 Petri 网的形式化定义。在此基础上，定义了一个封闭的 Web 服务合成算法，通过算法获得的框架能够对 Web 服务进行说明性的合成。进一步定义了一个自动 Web 服务合成算法，通过该算法，可以对 Web 服务进行有人工指导的半自动合成和无人工指导的自动合成。

Web 服务自动合成的研究还在进行中，下一步的工作主要是在 Web 服务的有色 Petri 网模型中引入语义 Web，利用 OWL-S 规范来实现服务发现、调用和合成的自动化。

参 考 文 献

[1] Tsalgatidou A, Pilioura T. An overview of standards and related technology in web services [J]. Distributed and Parallel Databases, 2002, 12(2/3):135-162

[2] Dustdar S, Schreiner W. A survey on web services composition [J]. International Journal of Web and Grid Services, 2005, 1(1):1-30

[3] Sun Haiyan, Wang Xiaodong, Zhou Bin. Research and Implementation of Dynamic Web Services Composition [J]. Lecture Notes in Computer Science, 2003, 2834:457-466

[4] Rao Jinghai, Su Xiaomeng. A Survey of Automated Web Service Composition Methods [J]. Lecture Notes in Computer Science, 2005, 3387:43-54

[5] Jensen K. An Introduction to the Theoretical Aspects of Colored Petri Nets [J]. Lecture Notes in Computer Science, 1994, 803:230-272

[6] Milanovic N, Malek M. Current Solutions for Web Service Composition [J]. IEEE Internet Computing, 2004, 8(6):51-59

[7] Hamadi R, Benatallah B. A Petri Net-Based Model for Web Service Composition [C]// Proceedings of the 14th Australasian Database Conference. Adelaide: Australia, Australian Computer Society, 2003:191-200

[8] Tan Z, Lin C, Yin H, et al. Approximate Performance Analysis of Web Services Flow Using Stochastic Petri Net [J]. Lecture Notes in Computer Science, 2004, 3251:193-200

[9] 郭玉彬，杜玉越，奚建清. Web 服务组合的有色网模型及运算性质[J]. 计算机学报，2006，29(7):1067-1075

[10] 钱柱中，陆桑璐，谢立. 基于 Petri 网的 Web 服务自动组合研究[J]. 计算机学报，2006，29(7):1057-1066

[11] Tan Z, Lin C, Yin H, et al. Approximate Performance Analysis of Web Services Flow Using Stochastic Petri Net [J]. Lecture Notes in Computer Science, 2004, 3251:193-200

[12] Narayanan S, McIlraith S. Analysis and simulation of Web Services [J]. Computer Networks, 2003, 42(5):675-693

[13] Ren Zhihong, Cao Jiannong, Chan A T S. Composition and automation of grid services [J]. Lecture Notes in Computer Science, 2003, 2834:352-362

[14] McIlraith S A, Martin D L. Bringing Semantics to Web Services [J]. IEEE Intelligent Systems, 2003, 18(1):90-93

[15] Paolucci M, Sycara K. Autonomous semantic web services [J]. IEEE Internet Computing, 2003, 7(5):34-41