一个基于偏序事件结构的 Web 服务模型及其形式化组装*)

韩婷婷 陈韬略 颜 峰 吕 建

(南京大学计算机软件新技术国家重点实验室 计算机软件研究所 南京 210093)

摘 要 随着 Internet 的发展,服务化和 Web 化趋势使得一个基于 Web 的分布式软件服务计算环境正在形成;同时随着大规模应用需求的不断涌现,单个的 Web 服务往往不能很好地满足一些复杂的应用。因此 Web 服务之间的集成组装就显得尤为重要。分析 Web 服务的交互和协同行为可以发现,分布性和并发性是基于软件服务分布计算系统的本质特征。这就往往使得组装的正确性难以得到保证,从而需要提供一定的手段加以支持;同时,基于软件服务分布计算系统的效率问题也值得关注。我们认为,利用偏序关系对这些问题加以处理是一种理想和有效的手段;而同时我们发现,在服务的组装中,服务与服务之间存在着一种自然的偏序关系。本文基于此,从形式化的角度研究了软件服务及其组装,提出了一种基于偏序事件多集的 Web 服务的形式化模型。本文从服务内部事件而不仅仅是服务的角度来考察并发问题,这使服务内和服务间的更多的事件可以并行执行,并使得组装后的系统可以更高效地进行实施,从而使得 Web 服务的并发粒度变细;在此基础上给出了一种服务组装语言和规则,以确保在组装过程中出现的局部变化不影响整体的 Web 服务的正确组装,从而能够从形式化的角度来规范 Web 服务的集成组装,使其正确性和效率能够得到保证。

关键词 Web 服务,偏序事件结构,服务集成组装

A Web Service Model Based on Partial Order Event Structure and its Formal Composition

HAN Ting-Ting CHEN Tao-Lue YAN Feng LU Jian

(State Key Lab for Novel Software Technology, Institute of Computer Software, Nanjing University, Nanjing 210093)

Abstract With the development of Internet, a software service computing environment based on Web has come into being. At the same time, as the requirement of large-scale applications, it is often not sufficient for single Web service to support some complicated applications. As a result, the composition of Web services becomes even more important. Due to the interaction and coordination between Web services, distribution and concurrency are two essential characteristics of the software service distributed system, which makes it difficult to ensure the correctness of the composition. Thus a mechanism supporting composition should be provided. Moreover, the efficiency of the software service distributed system is also worth considering. This paper investigates these problems from the point of formalism. Partial order relation, as an ideal and effective means, is utilized because of the natural partial order relation among Web services. Based on this, this paper gives a formal model of Web services based on partial order event multisets from the perspective of composition. We deal with the concurrency problem from finer grain of event inside the Web service itself other than the whole Web service, which makes it possible for more events in one Web service or from different Web services to execute concurrently and thus improve the efficiency. Furthermore, a Web service composition language is given with the rules that ensure an efficient composition when correctness is guaranteed. Some examples are invited to demonstrate the advantages of our model and composition approach.

Keywords Web services, Partial order event structure, Web service composition

1 引言

随着 Internet 的发展,计算机软件无论在外在形态、内在需求、关键技术还是在应用模式上都在经历着一场深刻的变革。其中一个显著的特点在于软件正由传统的复杂产品逐渐转变为简单易用的软件服务;进一步地,服务化和 Web 化趋势使得一个基于 Web 的分布式软件服务计算环境正在形成。同时,应用程序的开发模式也已经从过去的本地系统集成发展到更为复杂的多层系统。Web 服务结合了高效紧密的单层计算技术与面向消息的、松散耦合的 Web 技术,并通过集成提升了网络中众多应用程序的价值。简言之,Web 服务是一种自适应、自描述、模块化的新型 Web 应用程序,可以跨越 Web 进行发表、定位和调用,是封装成单个实体并发布到网络上以供其它程序使用的功能集合[1]。

分布性和并发性是基于 Web 服务计算系统的固有属性,这是因为随着大规模应用需求的不断涌现,仅靠 Web 服务本身有时并不能满足一些复杂的应用,这时 Web 服务之间的集成就是非常必要的了;同时,随着网络技术的不断成熟,特别是 Web 服务之间的交互能力的提高以及对 Web 服务的语义层面的研究与发展,也使得 Web 服务之间的集成成为可能。一般而言,Web 服务集成的研究主要经历了两个阶段:第一代的 Web 服务集成,本质上就是将已有的对象和组件(例如 CORBA、Java 或 COM) 实现为 Web 服务,即开发必需的 WSDL、SOAP 和其它 XML 文件,以及将 Web 服务绑定到中间层和后端的代码(如 Java)。然而,需要描述并实现的商务功能通常都包含了比单独存在的对象或组件多得多的业务逻辑,因此 Web 服务的粒度往往(也应该)比许多已有对象和组件的粒度更粗。基于这个原因,有了第二代 Web 服务集成。第二代 Web 服务集成,从本质上来讲是将已有的 Web 服务集成和组装成更为粗粒度的服务,Web 服务最大的潜能是一个集成平台。尤其是在商业过程中,由于商业流程的实施是一个庞大而复杂的过程,而且针对不同的企业,这个过程也存在较

*)基金项目:国家重点基础研究发展规划 973 项目(2002CB312002);国家自然科学基金(60273034);863 项目(2002AA116010);江苏省自然科学基金(BK2002203,BK2002409)。

大的差异，单个的 Web 服务无法满足企业级的需求。我们需要将各个不同的环节中的 Web 服务整合在一起，进而得到一个完整的流程[2]。

然而，作为分布式软件服务计算环境的关键技术之一，面向服务层次的服务组装（从动态的角度看体现为服务运行协同问题）依然是一个有待解决的研究课题。我们认为，目前对 Web 服务集成的研究还不够深入：已有的研究，如文[3,4]主要集中在为 Web 服务技术制定基于 XML 的标准，为 Web 服务集成定义原语（primitive）并使各 Web 服务间自动协作；在其基础理论方面的研究还停留在起步阶段。如何借助形式化的理论，为 Web 服务的集成建立系统模型，并在此基础上探讨 Web 服务集成的形式化描述、验证和效率等问题，是我们研究的核心问题。对这些问题加以深入的研究，对于 Web 服务的正确高效集成，从而实现大范围的跨企业实体的商务应用系统对接有着重要的理论和实际意义。本文主要针对此进行一些初步的探索。

在进行 Web 服务集成的过程中，集成的正确性和效率是两个我们需要关注的方面，简言之，我们需要在保证正确性的前提下尽可能地提高集成的效率。一般而言，集成效率的提高是通过挖掘 Web 服务之间的并发关系来实现的，而正确性是通过 Web 服务之间执行的顺序关系来保证。类比分布式数据处理中的相关问题和方法，我们可以发现为了保证并发事务的一致性和隔离性，要对事务进行并发控制，一般的方法是产生一个可串行化调度序列，在保证事务处理正确性的同时使得事务之间可以交叠甚至并行地执行，进而可以极大地提高并发度。我们对于 Web 服务的集成的研究思路受此启发，集成效率的提高可以通过使得更多的 Web 服务之间交叠或并行执行来实现，然而这必须建立在集成正确性的基础之上。而保证集成的正确性需要研究服务之间的集成顺序，这就很自然地让我们选择基于偏序关系的形式化机制来描述和验证 Web 服务的集成；同时，分布性和并发性是基于软件服务分布计算系统的本质特征，而以往并发领域的研究也告诉我们，通过偏序关系对此加以处理是一种理想和有效的手段：偏序关系可以很好地刻画这种并发性，并可以描述和发掘服务之间并发的可能性。任给两个服务，通过偏序关系可以判定其是否必须串行执行或可以并发执行。具体而言，服务之间的偏序关系可以帮助组装人员清晰地掌握服务的先后执行顺序，将那些必须串行执行的服务和执行的先后顺序无关的服务及服务序列区分开来，进而让更多的可以并发的服务及服务序列得以并行地执行。

进一步地，我们容易看到，服务与服务之间存在着一种自然的偏序关系。Web 服务的组装正是基于这种服务之间的偏序关系。更进一步地，我们认为这个偏序应该有两个层次，两种粒度。从服务的粒度上来说，这使得我们可以研究 Web 服务之间的并发关系。然而仅在服务层次上考虑并发是不够的，原因在于最终 Web 服务的实施是在事件的层次上的，不同服务的事件可以并发，同一服务内部的事件之间也可以并发，这对于 Web 服务的组装有着重要的意义。因此，我们从更细的事件的粒度来研究并发，这使得我们可以发掘服务内部的并发关系，从而极大地提高效率。

从 Web 服务的集成过程来看，有两方面需要我们重点关注，一是被集成的实体，二是要有一种集成语言。前者就是 Web 服务，而对于后者，我们提出了一种服务集成语言，根据不同的 Web 服务之间以及 Web 服务之内事件的偏序关系，采用语义上不同的集成组装方式对其进行组装。

基于以上的考虑，我们通过服务间的事件偏序关系来为 Web 服务建模，并在这个模型的基础上提出一种服务组装语言和规则对服务实施组装。组装语言描述的是 Web 服务之间通过语义上不同的组装方式（具体体现为组装子）进行组装，而组装规则则是为了保证在组装过程中事件和服务发生的变化并不会影响服务的正确组装。最后我们提出了从服务偏序到事件偏序之间的映射关系，我们发现在组装的过程中，在一些组装方式下可以获得更多的事件偏序的信息，从而更好地支持并发。这个模型和组装语言的意义在于一方面，可以帮助服务组装人员进行决策，通过模型的描述和集成规约，尽可能地避免集成过程中的一些错误；另一方面，可以更好地描述并发，使得尽可能多的服务能够同时进行，这样可以提高效率，节省时间，为整个实施过程提供一个宏观的指导。

本文的工作在于：第一，我们通过对偏序事件结构的考察，提出了一种形式化描述 Web 服务的模型；第二，我们提出了一种 Web 服务的组装语言；第三，基于这个组装语言给出了组装的规则以及从服务偏序到事件偏序的映射关系。

本文第 2 部分给出网络服务的直觉和形式化模型，并通过一个例子说明如何建立网络服务的模型；第 3 部分给出一个网络服务的组装语言，并给出了组装的规则；第 4 部分扩充了第 2 部分中的例子，并说明如何进行网络服务的组装；最后总结全文，并讨论进一步工作。

2 Web Service 形式化模型

2.1 Web 服务

Web 服务是指一些应用程序逻辑单元或代码，这些代码可以完成运算、数据库查询等工作，是以独立于平台的方式出现的[5]，它代表着可以从 Web 上存取的一个单位的业务、应用或系统功能。开发人员利用 Web 服务规范封装现有业务流程，并将其作为服务进行发布，也搜索和预订其他服务以及在企业内部与外部交换信息，从而将各种不同系统灵活地连接在一起。Web 服务使很多以不同编程语言开发的、由不同供应商提供的或运行在不同操作系统之上的企业软件之间可以互相通信。Web 服务广泛适用于任何 Web 环境，无论是在互联网、Intranet 还是在 Extranet，Web 服务均可用于 B2B、B2C 的应用中。

Web 服务的核心技术包括 XML、SOAP、WSDL 与 UDDI。然而，我们关注的角度在另一个方面，我们的目标是描述服务之间的并发。首先我们从服务的粒度出发，服务之间存在着偏序的关系；进而从事件的粒度上来讲，一个网络服务可以视为是一些事件的集合，由于服务的偏序，这些事件之间也存在偏序的关系，由服务的偏序可导出事件的偏序，因而我们可以从一个更细的粒度来描述并发，提高系统的效率[6]。

2.2 形式化模型

基于对以上 Web 服务的考虑，我们在本部分中提出一个形式化的模型。

定义 1 一个事件 $V=(Pt, MType)$ 是一个二元组，表示一个端口发送或者接受一个消息，其中 Pt 表示所有可能的端口的集合，$MType$ 表示所有可能的消息类型的集合。一个发送消息的事件呈形 $a!(m)$，a 是端口，$!$ 表示输出，m 是消息或实参；接受消息的事件呈形 $a?(x)$，类似地，a 是端口，$?$ 表示输入，x 是变量或形参，$m、x\in MType$。

定义 2 一个服务词汇表 F 是表示网络服务需要和提供的服务原语的集合，$F=F_p\cup F_r, F_p\cap F_r=\Phi, F_p$ 为网络服务 WS 所提供服务的词汇表，F_r 为网络服务 WS 所需服务的词汇表。

定义 3 一个事件偏序是一个四元组 $(V, F, \mu, \leqslant)$，其

中，一个事件集合 V，表示所有发生的事件；一个服务词汇表 F，表示网络服务需要和提供的服务原语(即事件类型)的集合；一个 V 上的偏序关系 $\leqslant$，满足反自反性、反对称性和传递性；一个标注函数 $\mu: V \to F$，将 F 中的符号赋给 V 中的节点，表示对应服务的事件的发生。由于同一个服务可以有多次事件发生，因此 μ 不必是单射。可在事件偏序[7]的基础上定义偏序事件多集[8]。

定义 4 一个偏序事件多集就是一个事件偏序的同构类，表示为 $[V, F, \mu, \leqslant]$。

由于 V 中不同事件被认为除标识外是完全同构的，并且可能被标注为相同服务原语，因此该偏序的定义域可看成一个多集，而不是一个集合。

定义 5 一个网络服务(Web Service)可被定义为一个二元组 $WS = (F, WSS)$，其中，F 为服务词汇表；WSS 为网络服务规约(specification)，描述该网络服务应当满足的规约或协议，WSS 是 F 上的正则表达式的集合。

我们选择正则表达式的目的在于正则表达式的表达能力可以部分剔除一些不合适的行为，为网络服务的行为划定一个范围，也就是说将正则表达式看作是一种行为模式，凡是符合正则表达式的网络服务的行为都可以认为是合法的行为。每一个正则表达式都表示一个网络服务可能产生的合法事件轨迹[9]。

2.3 Web 服务描述实例

在本节中，我们将给出一个实例来进一步说明如何用该形式化模型进行描述和建模。设想一个出货的商业流程，出货后可以选择陆路或水路的运输方式，当货运到目的地时就开始销售活动。

我们用三个网络服务来表示以上的三个活动：出货为 WS_1，陆路运输为 WS_2，销售为 WS_3。对 WS_1，$WS_1 = (F_1, WSS_1)$，$F_1 = F_{1p} \cup F_{1r}$，F_{1p} 为登记装箱 a，F_{1r} 为产品加工检验 b，WSS_1 为 $[V_1, F_1, \mu_1, \leqslant_1]$，其中 $V_1 = \{a!(m), a?(x), b?(x), b!(m)\}$，$a!(m)$ 是从端口 a 输出产品加工检验所需的参数，$a?(x)$ 是从端口 a 输入产品加工检验服务返回的结果，$b?(x)$ 是从端口 b 输入登记装箱服务提供的参数，$b!(m)$ 是从端口 b 输出提供给登记装箱服务的结果。对 WS_2 和 WS_3 也是类似的。图 1 是对该商业流程进行形式化建模的结果。

3 Web 服务组装

随着大规模应用需求的不断涌现，单个的 Web 服务在一些复杂的应用面前有时会显得能力不够，这时 Web 服务之间的集成组装就成为非常必须；同时网络技术的不断成熟，特别是 Web 服务之间的交互能力的提高以及对 Web 服务的语义层面的研究与发展，也使得 Web 服务之间的集成组装就成为可能。

从过程上来说，Web 服务的组装方式可以是各种各样的；从结果上来说，两个及两个以上网络服务组装可以认为是形成了一个新的 Web 服务。以下我们引入一个网络服务组装语言，我们称之为组合。组合是 Web 服务 WS 的集合，也就是一个 Web 服务与另一个 Web 服务之间协同交互的关系，进而可以形式地来描述 Web 服务的集成组装。

$WS_1 = (F_1, WSS_1), F_1 = F_{1p} \cup F_{1r}, WSS_1 = [V_1, F_1, \mu_1, \leqslant_1], V_1 = \{a!(m), a?(x), b?(x), b!(m)\}, \mu_1(a!(m)) = F_e, \mu_1(a?(x)) = F_{1r}, \mu_1(b?(x)) = F_e, \mu_1(b!(m)) = F_{1p}, a!\ m(\leqslant) a?(x), b?(x) \leqslant b!(m)$；

$WS_2 = (F_2, WSS_2), F_2 = F_{2p} \cup F_{2r}, WSS_2 = [V_2, F_2, \mu_2, \leqslant_2], V_1 = \{c!(m), c?(x), d?(x), d!(m)\}, \mu_2(c!(m)) = F_e, \mu_2(c?(x)) = F_{2r}, \mu_2(d?(x)) = F_e, \mu_2(d!(m)) = F_{2p}, c!\ m(\leqslant) c?(x), d?(x) \leqslant d!(m)$；

$WS_3 = (F_3, WSS_3), F_3 = F_{3p} \cup F_{3r}, WSS_3 = [V_3, F_3, \mu_3, \leqslant_3], V_1 = \{e!(m), e?(x), f?(x), f!(m)\}$，
$\mu_3(e!(m)) = F_e, \mu_3(e?(x)) = F_{3r}, \mu_3(f?(x)) = F_e, \mu_3(f!(m)) = F_{3p}, e!\ m(\leqslant) e?(x), f?(x) \leqslant f!(m)$；

图 1 形式化建模结果

3.1 组装语言

网络服务之间通过不同的操作相连，相应的服务词汇表 F 和网络服务规约 WSS 也将进行变更和合并，从而组合形成各种不同形态和行为的组合，而这些组合仍然是一个网络服务，并遵循一个网络服务规约。

定义 6 一个组合 $Z ::= WS \mid Z \to Z \mid Z + Z \mid Z \square Z \mid Z \diamond Z \mid Z^*$ 是 Web 服务 WS 的集合。其中，

(1) $Z_1 \to Z_2$ 表示两个进程之间的顺序连接，即当 Z_1 执行完就立刻执行 Z_2；

(2) $Z_1 + Z_2$ 表示两个进程之间的选择连接，即执行 Z_1 或 Z_2；

(3) $Z_1 \square Z_2$ 表示两个进程之间的并发连接，即同时执行 Z_1 和 Z_2；

(4) $Z_1 \diamond Z_2$ 表示两个进程之间的中断连接，即先执行 Z_1，当 Z_2 开始执行，立刻停止 Z_1 的执行；

(5) Z^* 表示一个进程的反复执行，可以认为是循环自连接。

由这些节点之间的操作组合可以描述出不同的服务组装行为。

我们称这些不同的组装方式为操作子，我们借鉴了程序设计语言中顺序、选择和循环这三种基本的执行方式；同时由于 Web 服务分布性和并发性的特点，参考了进程代数中有关并行算子的内容，以此来刻画并发；最后，中断算子的设计是从文[10]中得到的启发。

这些操作符之间的优先级是 * 优先级最高，$\to \diamond$ 其次，+ 再次，$\square$ 的优先级最低。我们将在第 4 部分通过一个具体的应用实例来说明服务的组装过程。

3.2 组装规则

下面，我们给出 Web 服务的一些组装规则。之所以在给出组装语言之后又给出组装规则，是因为在组装的过程中，服务中的一些事件多集可能会发生变化，如何确保在变化的情况下仍然能够正确地进行组装，以及几个 Web 服务在组装之后仍然是一个 Web 服务，并且仍满足一定的网络服务规约，为此，我们需要给出一组规则来约束和引导组装的正确进行。其规则如下：

(R1) $P = P_1 \to P_2$

$P_1 = (F_1, WSS_1), P_2 = (F_2, WSS_2)$；

其中，$WSS_1 = [V_1, F_1, \mu_1, \leqslant_1], WSS_2 = [V_2, F_2, \mu_2, \leqslant_2]$，$F_1 = F_{1p} \cup F_{1r}, F_2 = F_{2p} \cup F_{2r}$；

集成组装后：$P = (F, WSS), WSS = [V, F, \mu, \leqslant]$；其中

1) $F = F_p \cup F_r, F_p = (F_{1p} - F_{2r}) \cup F_{2p}, F_r = F_{1r} \cup (F_{2r} - F_{1p})$；其中 F_{2r} 和 F_{1p} 在语义上是针对于同一个服务的请求和提供方的词汇，通过 F_{2r} 和 F_{1p} 的匹配来进行顺序连接。

2) $V = (V_1 \cup V_2 - V_1 \cap V_2) \cup update(V_1 \cap V_2), update: V \to V$，这是可由组装者自定义的节点更新函数，对于顺序连接操作 $\to$ 前后的变化节点，$(V_1 \cap V_2)$ 是其公共的连接部分，这些节点对应的事件集合可能会由于连接发生改变，*update* 正是预先针对这些改变而采取的行为动作。

3)$\mu=\mu_1\cup\mu_2\cup\mu^-\cup\mu^+$,其中 μ^- 为负标注函数,μ^-:$(V_1\cup V_2-V_1\cap V_2)\rightarrow(F_1\cup F_2-F_1\cap F_2)$,该函数作用于在顺序连接操作前后失去定义的映射,其中$(V_1\cup V_2-V_1\cap V_2)$是没有改变的节点,$(F_1\cup F_2-F_1\cap F_2)$是原先作用在这些节点上的服务;$\mu^+$为正标注函数,$\mu^+$:$(V_1\cap V_2)\rightarrow(F_1\cup F_2)$,该函数作用于在顺序连接操作前后新增加定义的映射,其中$(V_1\cap V_2)$是变更过的新的节点,$(F_1\cup F_2)$是有可能作用在这些新增节点上的服务。

4)$\leqslant=\leqslant_1\cup\leqslant_2\cup(V_1\times update(V_1\cap V_2))\cup(update(V_1\cap V_2)\times V_2)$,其中 $V_1\times update(V_1\cap V_2)$表示 V_1 集合中的事件和 $update(V_1\cap V_2)$集合中事件可能产生的新的偏序关系,$update(V_1\cap V_2)\times V_2$ 表示 $update(V_1\cap V_2)$集合中事件和 V_2 集合中的事件可能产生新的偏序关系。

(R2)$P=P_1+P_2$

$P_1=(F_1, WSS_1)$,$P_2=(F_2, WSS_2)$;

其中,$WSS_1=[V_1, F_1, \mu_1, \leqslant_1]$,$WSS_2=[V_2, F_2, \mu_2, \leqslant_2]$,$F_1=F_{1\rho}\cup F_{1r}$,$F_2=F_{2\rho}\cup F_{2r}$;

集成组装后:$P=(F, WSS)$,$WSS=[V, F, \mu, \leqslant]$;其中

1)$F=F_\rho\cup F_r$,$F_\rho=F_{1\rho}\cup F_{2\rho}$,$F_r=F_{1r}\cup F_{2r}$;

2)$V=V_1\cup V_2$;

3)$\mu=\mu_1\cup\mu_2$;

4)$\leqslant=\leqslant_1\cup\leqslant_2$。

(R3)$P=P_1\square P_2$

$P_1=(F_1, WSS_1)$,$P_2=(F_2, WSS_2)$;

其中,$WSS_1=[V_1, F_1, \mu_1, \leqslant_1]$,$WSS_2=[V_2, F_2, \mu_2, \leqslant_2]$,$F_1=F_{1\rho}\cup F_{1r}$,$F_2=F_{2\rho}\cup F_{2r}$;

集成组装后:$P=(F, WSS)$,$WSS=[V, F, \mu, \leqslant]$;其中

1)$F=F_\rho\cup F_r$,$F_\rho=F_{1\rho}\cup F_{2\rho}$,$F_r=F_{1r}\cup F_{2r}$;

2)$V=(V_1\cup V_2-V_1\cap V_2)\cup update(V_1\cap V_2)$,$update$:$V\rightarrow V$,这是可由组装者自定义的节点更新函数,对于并发连接操作前后的变化节点,$(V_1\cap V_2)$是其公共的部分,在并发的过程中这些节点所对应的事件集合可能会由于连接发生改变,$update$ 正是预先针对这些改变而采取的行为动作。

3)$\mu=\mu_1\cup\mu_2\cup\mu^-\cup\mu^+$,其中 μ^- 为负标注函数,μ^-:$(V_1\cup V_2-V_1\cap V_2)\rightarrow(F_1\cup F_2-F_1\cap F_2)$,该函数作用于在顺序连接操作前后失去定义的映射,其中$(V_1\cup V_2-V_1\cap V_2)$是没有改变的节点,$(F_1\cup F_2-F_1\cap F_2)$是原先作用在这些节点上的服务;$\mu^+$为正标注函数,$\mu^+$:$(V_1\cap V_2)\rightarrow(F_1\cup F_2)$,该函数作用于在顺序连接操作前后新增加定义的映射,其中$(V_1\cap V_2)$是变更过的新的节点,$(F_1\cup F_2)$是有可能作用在这些新增节点上的服务。

4)$\leqslant=\leqslant_1\cup\leqslant_2$。

(R4)$P=P_1\diamond P_2$

$P_1=(F_1, WSS_1)$,$P_2=(F_2, WSS_2)$;

其中,$WSS_1=[V_1, F_1, \mu_1, \leqslant_1]$,$WSS_2=[V_2, F_2, \mu_2, \leqslant_2]$,$F_1=F_{1\rho}\cup F_{1r}$,$F_2=F_{2\rho}\cup F_{2r}$;

集成组装后:$P=(F, WSS)$,$WSS=[V, F, \mu, \leqslant]$;其中

1)$F=F_\rho\cup F_r$,$F_\rho=cut(F_{1\rho})\cup F_{2\rho}$,$F_r=F_{1r}\cup F_{2r}$;$cut(F_{1\rho})$是在 V_1 中断之间发挥作用的服务词汇集;

2)$V=cut(V_1)\cup V_2$,cut : $V\rightarrow V$,这是可由组装者自定义的节点中断函数,对于中断连接操作$\diamond$前后的变化节点,$cut(V_1)$是 V_1 中发挥作用的节点的集合,也就是在中断之前 V_1 中的节点集,组装者可通过 cut 函数对中断发生的条件和过程进行控制。

3)$\mu=\mu_1\cup\mu_2\cup\mu^-\cup\mu^+$,其中 μ^- 为负标注函数,μ^-:$cut(V_1)\rightarrow cut(F_1)$,该函数由于在中断连接操作前后可能会失去部分映射的定义,因此需要对某些部分重新映射,其中 $cut(V_1)$是没有改变的节点,$cut(F_1)$是原先作用在这些节点上的服务;μ^-为正标注函数,μ^+:$V_2\rightarrow cut(F_1)$,该函数由于在中断连接操作前后新增加映射的定义,其中 V_2 可能从中断前的 $cut(F_1)$中获取服务。

4)$\leqslant=\leqslant_1\cup\leqslant_2\cup(cut(V_1)\times V_2)$,其中 $cut(V_1)\times V_2$ 表示中断前 $cut(V_1)$的事件集合可能与 V_2 中的事件集合产生新的偏序关系。

(R5)$P=P_1^*$

$P_1=(F_1, WSS_1)$;

其中 $WSS_1=[V_1, F_1, \mu_1, \leqslant_1]$,$F_1=F_{1\rho}\cup F_{1r}$;

集成组装后:$P=(F, WSS)$,$WSS=[V, F, \mu, \leqslant]$;其中

1)$F=F_\rho\cup F_r$,$F_\rho=F_{1\rho}$,$F_r=F_{1r}$;

2)$V=V_1\cup update(V_1)$,$update$: $V\rightarrow V$,这是可由组装者自定义的节点更新函数,对于循环自连接操作 * 前后的变化节点,在自连接的过程中这些节点所对应的事件集合可能会由于连接发生改变,$update$ 正是预先针对这些改变而采取的行为动作。

3)$\mu=\mu_1\cup\mu^+$,其中 μ^+为正标注函数,μ^+:$V\rightarrow F_1$,该函数作用于在循环自连接操作前后新增加定义的映射。

4)$\leqslant=\leqslant_1\cup(update(V_1)\times V_1)\cup(V_1\times update(V_1))\cup(update(V_1)\times update(V_1))$。

3.3 偏序的重构

在某些组装操作下,如顺序、中断或循环自连接,服务之间的偏序是已知的。事实上,在决定组装服务操作之前我们可以知道这些服务之间的依赖关系。我们在服务偏序的基础上可以推导出在事件集 V 上的偏序关系$\leqslant$,研究$\leqslant$是为了从更细的粒度上描述和验证网络服务之间的集成,从而使得描述更为精确、验证更为高效。一般地,我们假设在一个服务内仅当所有向外请求的服务都满足后才能向外提供服务。以下给出服务偏序到事件偏序之间的映射关系。

(1)$\dfrac{P_1\rightarrow P_2}{a!\ (m)\leqslant b?\ (x)}$,$a$, b 属于不同的节点;

(2)$\dfrac{-}{a?\ (x)\leqslant a!\ (m)}$;

(3)$\dfrac{WS_1\rightarrow WS_2\rightarrow WS_3}{a_{2r}?\ (x)\leqslant b_{2\rho}!\ (m)}$,$a$ 属于同一个节点,且 a_{2r}属于 F_{2r},$a_{2\rho}$属于 $F_{2\rho}$。

为了更清晰地阐述这三条规则,我们如图 2 所示,$WS_1\rightarrow WS_2\rightarrow WS_3$,其中(1)描述的是两个网络服务之间事件的偏序关系,即发送消息事件必在接收消息事件之前发生;(2)(3)描述的是一个服务内事件之间的偏序关系,(2)表示接收服务请求消息的事件必在返回结果消息事件之前发生,(3)表示接收向外请求服务 F_r 结果事件必在提供向外服务 F_ρ 的返回结果之前发生。

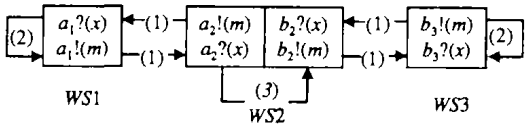

图 2 服务偏序到事件偏序的映射关系

4 网络服务组装实例

上一节中我们介绍了一个网络服务的组装语言以及组装规则，为了进一步说明网络服务的组装过程，我们将扩充 2.3 中的应用实例，以说明这些 Web 服务之间的交互与集成关系。图 3 给出的是一个被简化的商业流程，实际运行中的流程更为复杂，我们择其要点加以说明。

当一个公司接到订单 WS_{order} 后，将会买原料 $WS_{material}$ 并进行生产加工 $WS_{produce}$，但出于资金流动以及效率的考虑，不会一次将所有的原料买齐然后堆积在生产线上等待生产加工，因此进原料和加工这两个服务是循环进行的。

当生产完成后，将会安排出货 $WS_{shipment}$。根据不同的目的地会选择不同的运输方式飞机运输 WS_{plane}，火车运输 WS_{train}，轮船运输 WS_{ship}，这些方式之间是通过选择连接的。

运输到目的地后便可开始销售，公司的策略是一旦新型号的商品上市 $WS_{newsale}$，便停止旧型号商品的销售 $WS_{oldsale}$。销售完成后还要进一步跟踪，并实施售后服务 WS_{after}。

与此同时，另一条流程也在进行中。当接到订单时，公司便着手开始为新商品进行造势的宣传广告 WS_{prope}；当商品完成生产准备上市时，便进一步铺开广告攻势，使新产品能迅速在大范围内大规模推广 WS_{spread}；当商品的销售达到一定程度时，公司应适当推出代表公司而非产品本身的形象广告 WS_{image}，并以此使更多的人接受这个公司以及其更多的产品。

在这样的一个应用背景下，我们用 2 中的模型来描述这个流程，如下：

$$P=(WS_{order}\ \square\ WS_{prope})\rightarrow(WS_{material}\rightarrow WS_{produce})^{*}\rightarrow (WS_{shipment}\ \square\ WS_{spread})\rightarrow(WS_{plane}+WS_{train}+WS_{ship})\rightarrow(WS_{oldsale}\diamond WS_{newsale})\rightarrow(WS_{after}\diamond WS_{image})$$

本文的目的并非是研究如何从偏序关系中找出最多的并发事件，而是关注如何从组装的过程中建立偏序关系。限于篇幅，我们只给出组装的步骤而省略了其中由偏序关系挖掘并发事件的细节（比如 *update* 和 *cut* 或 μ^{-} 和 μ^{+} 函数的定义等），具体的步骤如下：

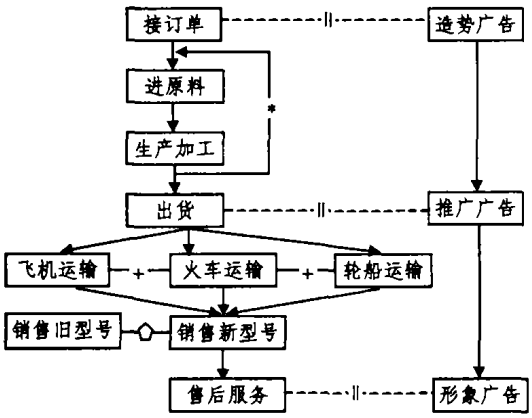

图 3 简化的商业流程

第一步，检查每两个待组装服务的服务词汇是否匹配，事件集是否完整；

第二步，用 *update* 和 *cut* 函数描述组装后事件多集和服务词汇表发生的一些变化，用 μ^{-} 和 μ^{+} 函数对变化的映射标注进行修正；

第三步，由偏序的传递，找出所有潜在的事件的偏序；

第四步，从事件的粒度找出尽可能多的并发事件。

结论及进一步工作 Web 服务的迅速发展与成熟使得一个基于 Web 的分布式软件服务计算环境正在形成。Web 服务结合了高效紧密的单层计算技术与面向消息的、松散耦合的 Web 技术，集成并提升了网络中众多应用程序的价值。然而仅靠 Web 服务本身有时并不能满足一些复杂的应用，比如商业应用等，这就亟需将已有的 Web 服务集成和组装成更为粗粒度的服务，以满足更多的需求。

本文就是在此应用背景下，从形式化的角度对 Web 服务的集成和组装进行研究。具体而言，我们首先从服务的偏序关系入手，研究 Web 服务之间的并发性，因为服务之间的偏序关系可以帮助组装人员清晰地掌握服务的先后执行顺序，进而让更多的可以并发的服务并行地执行；继而我们从更细的事件的粒度来考察服务的并发，使得不同服务的事件可以并发，同一服务内部的事件之间也可以并发，从而更大地提高并行度。进一步地，组装语言和组装规则的提出，给组装过程提供了一个保证，使得在可以组装的情况下，不会因为一些局部的变化影响整个组装的正确性。最后服务偏序到事件偏序之间的映射关系能在一些组装方式下获得更多的事件偏序的信息，从而更好地支持并发。

本文的工作主要在于通过服务间的偏序关系来为 Web 服务的集成建立形式化的模型，并提出了一个网络服务的集成语言和集成规则对服务进行组装。这个模型和组装语言的意义在于通过模型的描述和集成的形式化规约可帮助服务组装人员进行决策，从而避免集成过程中的可能的一些错误；同时，基于偏序关系的模型可以更好地描述并发，使得尽可能多的服务能够同时进行，这样可以提高效率，并为整个实施过程提供一个宏观的指导。

进一步的工作在于我们可以利用模型检测技术，研究该模型自动地检查服务集成的错误的问题；另外，对服务词汇的语义研究上也可以进行深入的探索。

参 考 文 献

1 Tidwell D. Web services: the web's next revolution. [http://www-106. ibm. com/developerworks/edu/ws-dw/wsbasics-i. html]

2 ITU2T/ SG16/ VCEG (Q. 6). H. 26L Test Model Long-Term Number 8 (TML - 8). Document VCEG- N10, Video Coding Experts Group (VCEG) 14th meeting, Santa Barbara, CA, USA, Sep. 2001. 24～27

3 Leymann F. Web services flow language (WSFL 1. 0). [http://www23. ibm. com/software/ solutions/webservices/pdf/WSFL. pdf], 2001

4 Thatte S. XLANG: Web services for business process design. [http://www. gotdotnet. com/ team/ xml - wsspecs/xlang2c/default. htm], Microsoft Corp., 2001

5 Specification: Business Process Execution Language for Web Services Version. [http://www. ibm. com/developerworks/library/ws-bpel/]

6 Uffe Henrik Engberg. Partial Orders and Fully Abstract Models for Concurrency, 1990

7 Pratt V. Modeling concurrency with partial orders. Int J of Parallel Progra, 1986, 15(1): 33～71

8 Grumbach S, Milo T. An Algebra for Pomsets. In : proc. of ICDT'95, Springer-Verlag, 1995. 191～207

9 Nierstrasz O. Regular types for active objects. ACM SIGSOFT Notes, 1993, 18(1): 115

10 任洪敏，钱乐秋. 构件组装及其形式化推导研究. 软件学报，2003，14(6)：1066～1074