一种支持 Web 服务运行时适应性的框架

段辉映[1] 杨 丹[2] 吴映波[3]
(重庆大学软件工程学院 重庆 400044)[1] (重庆大学计算机学院 重庆 400044)[2]

摘 要 本文对Web服务的运行时适应性进行了研究。首先,文章对当前的研究现状以及存在的问题进行了分析。然后,提出了一种完整的支持Web服务运行时适应性的框架。该框架在原有的发现和绑定机制基础上增加了前提条件的约束,提高了过程的执行效率。同时对BPEL和WSDL语言做了改进以支持语义Web服务选择。最后给出了该框架的具体实现方式。

关键词 Web服务组合,运行时适应性,发现和绑定机制

A Framework for Runtime Adaptability of Web Service

DUAN Hui-Ying[1] YANG Dan[2] WU Ying-Bo[3]
(Institution of Software Engineering, Chongqing University, Chongqing 400044)[1]
(Institution of Computer, Chongqing University, Chongqing 400044)[2]

Abstract This paper is a research on the run time adaptability of Web service. First of all, it analyses the current research state and problem. Then it proposes an intact framework for run time adaptability of Web service. Based on the ordinary find and bind mechanism, the framework adds the precondition to improve the performance of process execution. At the same time, it has an improvement on BPEL and WSDL to support semantic Web service selection. At last, the paper gives a detail implementation.

Keywords Web service composition, Run time adaptability, Find and bind mechanism

1 引言

Web服务正从概念的普及向规模化应用发展。最初提出了由服务提供者、服务访问者和服务中介三者构成的Web服务体系架构、Web服务的描述语言WSDL和传输协议SOAP。到目前为止,全球已经建立了相当数量的Web服务资源。如何将现存的原子性Web服务更合理的组织起来形成更复杂的Web服务以及如何更充分地实现透明的互操作成为目前研究的重点。Web服务组合语言和语义Web服务就是为了解决上述问题而产生的[1]。在此基础上,如何进一步提高Web服务互操作的灵活性则涉及到Web服务运行时适应性问题。如何使对Web服务的调用更加的灵活和机动?如何使面向服务的分布式系统可以适应软硬件环境的变化?我们需要在运行时动态的调用Web服务。

2 Web服务运行时适应性的研究现状与存在的问题

Web服务组合语言[2]的出现,给Web服务适应性的研究提出了更高的要求。科研人员从Web服务组合语言与工作流技术的比较中发现两者的相似之处。它们都是对过程的一种描述。Web服务组合借鉴了工作流的概念,通过XML的方式定义了控制流、数据流、活动和参与者等元素。但它们也存在不同。传统的工作流的生命周期被简单的划分成构造时(build time)和运行时(run time)。在此基础上,文[3]提出了Web服务组合的生命周期模型。其中包括:过程模板建模和集成阶段、过程定义产生、编译时、预处理阶段、部署阶段、运行阶段和运行后期。文[3]中还首次提到一种称为发现和绑定(find and bind)子阶段的概念。它是运行阶段的一部分,用来实现Web服务的选择和动态调用。文[4]提出建立Web服务元模型以实现服务流程的具体实现和流程本身的分离。同时它给出了实现Web服务适应性的方法的一种分类。针对其中的Web服务实例(WS instance)、服务类型(service type)和过程逻辑(process logic)提出了BPEL的改进方法。文[5]进一步完善了发现和绑定机制,将其细化为查找、选择和绑定三个阶段,如图1所示,并针对Web服务实例给出了一个比较详细的实现。

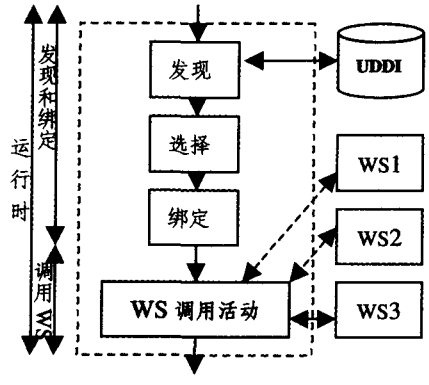

图1 发现和绑定机制

目前对Web服务运行时适应性的研究还存在不少问题。首先,Web服务选择策略的需要形式化的定义。其次,发现和绑定机制在执行性能方面需要充分考虑。再次,需要把语义在Web服务适应性中的作用体现出来。第3节所提出的框架就是为了解决上述问题。

3 支持Web服务运行时适应性的框架

3.1 发现和绑定机制

本文在原有的发现和绑定机制上,加入了"前提条件"的概念。"前提条件"体现了Web服务发现和绑定机制对初始

段辉映 硕士研究生,主要研究方向:Web服务,软件过程、软件度量;**杨 丹** 教授,博士,主要研究方向:软件过程、软件质量、软件项目管理;**吴映波** 讲师,博士研究生,主要研究方向:软件过程技术与管理,现代软件质量工程,供应链管理与技术。

环境的刻画。例如，实例化一个 BPEL(Business Process Execution Language)实例时需要经历完整的发现和绑定过程。但如果该实例存在，执行同一条〈invoke〉语句时就没必要经历整个发现和绑定的过程了，除非 BPEL 引擎发现〈invoke〉对应的选择策略发生了变化。这就是"前提条件"的作用。图 2 详细阐述了发现和绑定机制。BPEL 引擎是整个框架的控制中心，它根据不同的前提条件执行不同的发现和绑定操作。策略/绑定表是记录选定的策略和具体绑定信息的数据表。它的生命周期是应用级的，不会随着一个过程实例的消亡而消失。不同的前提条件对应的发现和绑定操作都会对该表进行读/写操作。策略表存放了该过程所有 Web 服务调用点的选择策略信息，可随时更新。

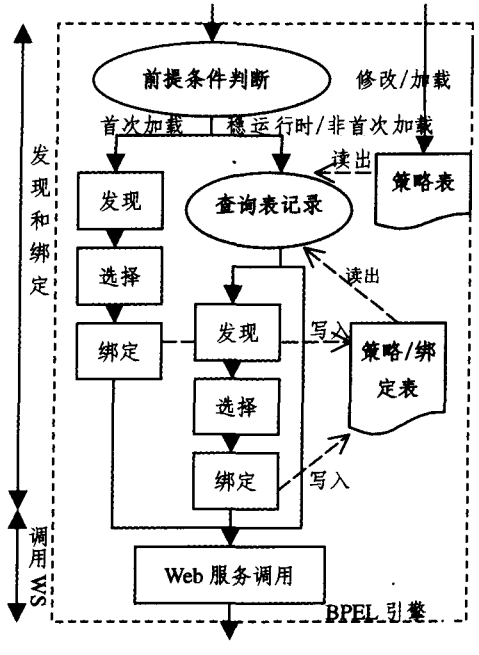

图 2　改进的发现和绑定机制。其中，发现的过程是从 UDDI 中获得候选服务的过程

运行时适应性涉及的前提条件有三种：首次加载运行、非首次加载运行和稳运行时。首次加载运行指一个部署后的 BPEL 文件头一次在某个 BPEL 容器里运行，此时需向策略/绑定表写入绑定信息。非首次加载运行是和首次加载运行相对应的概念。区分这两个概念，是因为对于一些选择策略相对固定的 BPEL 过程，可以在非首次加载运行时读取首次加载运行后留下的记录，不需要重新发现和绑定。稳运行时指在整个运行时除开前两个时期的其它时间跨度。此阶段的运行效率是最高的。稳运行时可以充分使用策略/绑定表的信息，使与 UDDI 通信量最小化。

策略/绑定表是实现运行时适应性的核心元素。它的字段包括：选择策略 ID、BPEL 过程实例 ID、位置信息、BPEL 过程 ID 和绑定信息。选择策略 ID 存放当前过程实例中〈invoke〉的选择策略标识符。BPEL 过程实例 ID 用来标识某一个过程实例。BPEL 过程 ID 用来标识 BPEL 过程文件。这两个标识符用来确定一个唯一的过程或者过程实体。当 BPEL 容器中不存在该过程的实例时，可以根据 BPEL 过程 ID 来记录该过程的选择策略历史。这样可为非首次加载运行提供必要信息。BPEL 引擎如果发现存在这样的记录，同时又未发现选择策略的改变，就沿用其绑定信息。

策略表中存放的是选择策略的集合。选择策略是 Web 服务选择的判断条件。下面是选择策略的形式化定义：

定义 1(Web 服务)　定义 W 为 Web 服务的集合。w 为具体的属性，使 $w \in W$。

定义 2(属性与属性值)　定义 A(Attributions)为某个 Web 服务的属性集合。a 为具体的属性，使 $a \in A$。一个具体的 Web 服务 w 对应的属性可以表示为 a_w。定义 v(value)为 a 的一个有效取值。

定义 3(操作符)　定义 ro 为关系操作符号，其中包括〈，=，〉，〈=，〉=等。定义 lo 为逻辑操作符，其中包括与，或和非，定义 LO 为一个逻辑操作符序列。

定义 4(选择策略函数)　定义函数 sp(Select Policy)，由前三个定义有 $sp(W,a_w,v,ro) \subseteq W$。令 $n=|LO|$：

(1)$n=1$，该函数为简单选择策略函数，有等价函数 $sp(W,a_w,v,ro)$。

(2)$n>1$，该函数为复杂选择策略函数，有等价函数 $SP(sp(W,a_1,v_1,ro_1),\cdots,sp(W,a_{|LO|+1},v_{|LO|+1},ro_{|LO|+1})),SP \subseteq W$。

复杂选择策略函数 SP 是对 n 个简单选择策略函数 sp 的集合运算。求解复杂分组函数的过程分两步：首先求出所有 sp 的结果集，然后对所有结果集进行集合运算得到最终的集合。

属性是语义相关的，也就是说 BPEL 选择策略文件的创建者和 Web 服务的语义描述文件的创建者共享一个语义规范。这个规范可以使用权威的都柏林核心[6]。例如某个 Web 服务的提供者使用了 date 这个术语来描述该服务有效期限。那么，调用方在定义选择策略时也可以对 date 进行语义限制。目前，适用于 Web 服务的语义元素还很少，现有的元素对 Web 服务的语义信息支持也不够，这方面规范的研究需要大大加强。

对应 BPEL 中的一个 Web 服务调用点，有且仅有一个选择策略。一个 BPEL 过程的所有选择策略都放在一个名为 *.sp 的文件中。文件随对应的 BPEL 过程实例一起加载到 BPEL 容器中。管理员可以随时(编译时/运行时)对该文件进行修改。

sp 文件由若干选择策略组成。选择策略之间由";"分隔。每条选择策略由三部分组成，按从左至右的顺序分别是：选择策略 ID、冒号和选择策略函数。其中，选择策略 ID 是 BPEL 文件中 Policy_table 属性的值。例如：Query：date>"2006/12/31"&& modified>"2005/12/01"。Query 是选择策略 ID。date="2006/12/31"&& modified="2005/12/01"的含义是指该 Web 服务的有效期限必须大于 2006 年 12 月 31 号，同时该 Web 服务的更新日期必须大于 2005 年 12 月 1 号。只有满足这个语义判断条件的 Web 服务才会被选中。

3.2　扩展的 BPEL

BPEL 中针对 Web 服务调用的标签只有〈invoke〉。对 BPEL 的扩展只需要对〈invoke〉进行扩展即可。其描述如下：

```
〈invoke partnerLink="NCName"
        portType="QName"?
        operation="NCName"
        inputVariable=" NCName"?
        outputVariable="NCName"?
standard-attributes〉
  〈find_bind policy_table=" NCName"〉?
〈! –detail definition–〉
〈/invoke〉
```

改进的 BPEL 语言在〈invoke〉语句中嵌入了〈find_bind〉标签。该标签有属性 policy_table，用来存放关联的策略表的引用。

3.3　扩展的 WSDL

WSDL[8]的扩展工作主要体现在其携带语义信息能力的增强方面。其描述如下：

(下转第 144 页)

D'，在处理步骤(3)时，对每一个 d_i 都建立了一个 Q_i，Q_i 是 d_i 的所有对外相交的属性集，即 $W_{N'}(d_i)=Q_i$，故 d_i 可归约至 Q_i。在步骤(4)把所有的 Q_i 并成了属性集 K，因而每一个 $d_i \in D'$ 所产生的关系模式对应的超图可归约至由 K 所产生的关系模式对应的超图中，最后只剩下由 K 所产生的关系模式 R_k，可归约至空，故该模式分解 ρ 具有无 α 环性。证毕。

3 算法、时间复杂度

根据定理5，完全可以给出，当 F 有内部冲突时，$R\langle W,F\rangle$ 分解为满足 P_2 且具有无 α 环的模式分解算法。

算法 1 Q-test(测试 D 是否满足条件 $\sum_1$ 和 $\sum_2$)

输入：D；
输出：若 D 同时满足条件 $\sum_1$ 和 $\sum_2$，输出 false；否则，输出 true；
login
  for 每一个 $d_i \in D, X_i^k \in L_i, A \in FR(X_i^k)$ do // D 若不满足条件 $\sum_1$ 输出 ture//
    if 存在 $d_j \in D(i \neq j)$ and $X_i^k \subseteq W_j$ then return；
  $D' := D$；
    for 每一个 $d_i \in D'$ do
      if 存在 $d_k \in D'$ and $W_i \cap (\bigcup_{j \neq i} W_j) \subseteq W_k$ and $d_j \in D'$ then
        $D' := D' - \{d_i\}$；
  for 每一个 $d_i \in D$ do // D 若满足条件 $\sum_2$ 输出 ture//
      if $X_i^k \in L_i, A \in FR(X_i^k)$ and $X_i^k A \subset \bigcup_{j \in N} W_{N'}(d_j)$ then
        return (F)

该算法的时间复杂度显然为 $O(n^2p)$。

算法 2 FJ_α(F 有内部冲突、满足保持 FD、$BCNF$ 且无 α 环的分解)

输入：$R\langle W,F\rangle$；
输出：$\rho=\{R_1,R_2,\cdots,R_n\}$；
login
  $D := GB\text{-}Set(F)$[3] //求出 F 的归异依赖集//。
  $D := Mini\text{-}GB(D)$[3] //求出 D 的归异依赖集//。
  if Q-test(D) then return
  $D' := D$；
  for 每一个 $d_i \in D'$ do
    If 存在 $d_k \in D'(i \neq k)$ and $W_i \cap (\bigcup_{j \neq i} W_j) \subseteq W_k$ and $d_j \in D'$ then
    $D' := D' - \{d_i\}$；
  $Q := \{\emptyset\}$；$Q_i := \emptyset$；//对每一个 $d_i \in D'$ 建立一个属性集 Q_i//
    for 每一个 $d_i \in D'$ do
      $Q_i := W_i \cap (\bigcup_{j \neq i} W_j)$；$Q = Q \cup \{Q_i\}$；
  $K = \emptyset$；
    for 每一个 $Q_i \in Q$ do
      $K := K \cup Q_i$
    $P := \emptyset$；
  for 每一个 $d_i \in D'$ do
      将 W_i 构成一个关系模式 R_i；$\rho := \rho \cup R_i$；
    将 K 构成一个关系模式 R_k；$\rho := \rho \cup R_k$；
  for 每一个 $R_i \in \rho$ do //删除被其它关系模式包含的 R_i//
    if 存在 $R_i (i \neq j)$ and $R_i \subseteq R_j$ then
      $\rho := \rho - \{R_i\}$
return(ρ)

显然，根据定理5该算法是正确的。如果设 n 为输入的属性个数。p 为 FD 的个数，算法的第(1)、(2)、(3)步的时间复杂度分别为 $O(np)$、$O(p^3)$、$O(n^2p)$，其余各步均小于 p，故该算法的复杂设为 $O(p^3+n^2p)$。

参 考 文 献

1 郝忠孝. 无内部冲突满足 P_3 的无 α 环的数据库模式分解Ⅰ：分解的基本理论. 计算机研究与发展，1998，35(4)：301～304
2 郝忠孝，等. 数据库模式分解为满足 P_3 及无 α 环的条件. 计算机研究与发展，1999，36(1)：101～105
3 郝忠孝，等. 最小归并依赖集求解的算法. 计算机研究与发展，1997，34(增刊)：267～270
4 郝忠孝，等. 函数依赖集 F 有内部冲突的判定问题的研究. 计算机研究与发展，2004，41(9)：1540～1544
5 郝忠孝，等. 有内部冲突的 F 的广义左、右部冲突判定算法. 计算机研究与发展，2004，41(11)：1924～1929

(上接第 123 页)

```
〈wsdl:service name="nmtoken"〉 *
    〈wsdl:documentation … /〉?
        〈wsdl:port name="nmtoken"
                   binding="qname"〉 *
    〈wsdl:documentation … /〉 ?
    〈soap:address location="uriReference "/〉
          〈sa:attribute〉 *
              〈sa:id/〉
              〈sa:value/〉
          〈/sa:attribute〉
          〈- - extensibility element - -〉
          〈/wsdl:port〉
          〈- - extensibility element - -〉
〈/wsdl:service〉
```

改进的 WSDL 在〈port〉元素中加入了〈sa:attibute〉元素。该元素由 ID 和 value 组成。〈sa:attibute〉元素允许有多个，每一个〈sa:attibute〉描述了该 Web 服务的一项语义信息。

所有的语义 Web 服务描述语言，包括 OWL-S，都要关联到 WSDL，供 Web 服务的访问者查询和发现。这种从语义描述语言到 WSDL 的转换可以通过直接扩展 WSDL 的方式实现，也可以使用 XSLT 文件实现。本框架只是对 WSDL 进行扩展，所以可适用于任何语义 Web 服务的调用框架。

4 框架的实现

该框架的核心部分是 BPEL 引擎。现存的 BPEL 引擎有很多，包括 IBM[9] 和 Oracle[10] 的产品。但它们都过于庞大，而且也没有相应的源码。本文推荐使用 ActiveBPEL[11]。它是一个开源的 java 项目。对于 BPEL 的改进可以在此基础上进行。同时，为了充分沿用 ActiveBPEL 已有功能，减少开发量，本文建议使用面向方面的编程模式(AOP)，具体的开发框架可以使用基于 eclipse 的 AspectJ 插件。

UDDI 的选择，本文建议使用 SAP 的 UDDI 注册机构[12]。它使用的是最新的 UDDI V3。

结束语 本文对 Web 服务的运行时适应性的研究现状做了全面分析，提出了一个支持 Web 服务运行时适应性的框架。该框架进一步完善了发现和绑定机制，对 BPEL 和 WSDL 提出了具体的改进方式以支持带语义的 Web 服务选择。由于语义 Web 服务的相关理论还在不断完善之中，该框架对带语义的 Web 服务选择的支持还没达到理想的程度，希望能在今后的研究工作中不断改进。

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
2 饶元，冯博琴，李尊朝. 基于 Web Services 的服务合成技术研究综述. 系统工程与电子技术，2005，8：1481～1489
3 Karastoyanova D, Buchmann A. Development Life Cycle of Web Service-based Business Processes. Enabling dynamic invocation of Web services at run time. In: Proceedings of The 2nd International Workshop on Web Services: Modeling, Architecture and Infrastructure 2004 (WSMAI-2004), April 2004
4 Karastoyanova D, Buchmann A. Extending Web Service Flow Models to Provide for Adaptability. In: Proceedings of OOPSLA '04 Workshop on "Best Practices and Methodologies in Service-oriented Architectures: Paving the Way to Web-services Success", Vancouver, Canada, 2004
5 Karastoyanova D, Houspanossian A, Cilia M, Leymann F, Buchmann A. Extending BPEL for Run Time Adaptability. In: Proc. of the 9th International Enterprise Distributed Object Computing Conference (EDOC 2005), Enschede, The Netherlands, September 2005
6 Dublin Core Metadata Initiative. http://dublincore.org/
7 Curbera F, Goland Y, Klein J, Leymann F, Roller D, Thatte S, Weerawarana S. Business Process Execution Language for Web Services (BPEL4WS) 1.1, May 2003
8 W3C: Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language, W3C Candidate Recommendation, 2006. http://www.w3.org/TR/2006/CR-wsdl20-20060327/
9 IBM developerWorks: WebSphere Business Integration Server Foundation Process Choreographer. http://www-106.ibm.com/developerworks/Websphere/zones/was/wpc.htm
10 Oracle Corporation: Oracle BPEL Process Manager 2.0. 2004
11 ActiveBPEL Engine. http://www.activebpel.org/
12 SAP UDDI Business Registry. https://uddi.sap.com/