基于生命周期的Web服务合成架构研究

杨 丹[1] 申德荣[2]
(辽宁科技大学软件学院网络工程系 鞍山 114051)[1]
(东北大学信息科学与工程学院计算机软件与理论研究所 沈阳 110004)[2]

摘 要 Web服务合成使Web服务的重用和组装以及提供增值服务成为可能。提出一个完善、有效的Web服务合成系统架构对于Web服务合成的发展和研究有着重要作用。根据Web服务合成生命周期的3个阶段,提出和讨论了一个基于Web服务合成生命周期的系统架构。该架构支持合成服务流程的建模、合成和执行调用。在建模阶段,使用可视化的流程定义器,经过图形解释器和定义语言文档解析后存入服务仓储。在合成阶段,考虑到异构问题引入基于本体的服务匹配器,还考虑到合成服务的服务质量(QoS)管理,因此使用基于多值背景值依赖的方法来查找符合用户偏好的服务。在执行阶段为了保证合成服务的正确执行,提供了监控器、异常处理器和事务适配器。

关键词 Web服务合成,生命周期,架构,服务质量
中图法分类号 TP391 **文献标识码** A

Web Service Composition Architecture Based on Life Cycle

YANG Dan[1] SHEN De-rong[2]
(Department of Network Engineering, Software College, University of Science and Technology Liaoning, Anshan 114051, China)[1]
(Institute of Computer Software and Theory, College of Information Science and Engineering, Northeastern University, Shenyang 114000, China)[2]

Abstract Web service composition makes it possible for Web service reused, assembled and providing value-added service. It is important to propose a complete, validate architecture for development and research of Web service composition. A Web service composition architecture based on the life cycle was proposed. The architecture supports process modeling, composition and execution. At the phase of modeling a GUI tool was provided to define the process which then was parsed by document parser to be stored in the service storage. In the composition phase services matcher based on ontology was introduced to solve service heterogeneous problem, and QoS of composite service was also considered. In the execution phase process monitor, exception handler and transaction adapter were provided.

Keywords Web service composition, Life cycle, Architecture, QoS

1 引言

1.1 Web服务合成

由于高灵活性和对Web服务资源的重用性,Web服务合成的重要性已经被广泛认识。关于合成服务的规范、系统架构等相关研究也是业界和学术界的热点问题。Web服务合成需要考虑多方面的问题,如合成服务建模问题、查找匹配服务问题、异构问题、动态合成问题、执行监控问题、分布式事务问题等。目前的Web服务合成的研究大多是在传统的工作流集成的研究之上进行,利用工作流中的管理方法合成Web服务。但是不同于传统的工作流系统,Web服务合成系统有其自身特点和待解决的问题。因此提出一个完善、有效、合理的Web服务合成系统架构,对于Web服务合成的发展和研究有着重要的作用。

1.2 相关工作

eFlow[1]系统提供定义、建立和监控合成服务的功能。eFlow支持动态提供者选择,提出了多服务节点、通用节点的概念,在事务处理上提出了事务区的概念。SELF-SERV[2]原型系统采用五层体系结构:服务层、会话层、目录层、通信层、用户层。它提出服务容器的概念,使用对等的服务执行模型。存在的问题是参与合成服务的服务提供者必须安装该系统提供的一个协调器和包装器来实现对等的(peer-to-peer)执行。这就存在着安全问题,而且如果不安装这两个程序的服务,提供者就不能参加服务合成,从而在一定程度上限制了该系统的应用范围和领域。WebTransact[3]系统采用多层的体系结构,分为远方服务层、协调层、合成层。通过扩展WSDL提出了基于XML的Web服务事务语言和2L-guaranteed-termination的事务模型。这些服务原型系统提出的架构各有侧重点,但是都没有从Web服务合成的生命周期考虑。

2 Web服务合成系统架构概述

2.1 Web服务合成的生命周期

Web服务合成按其生命周期大体可以分成3个阶段:建模阶段、合成阶段、执行阶段。在建模阶段,使用者或设计者

到稿日期:2009-05-06 返修日期:2009-07-16 本文受国家863项目(2008AA01Z146)资助。
杨 丹(1978—),女,讲师,主要研究方向为分布式数据库、Web服务合成等,E-mail:asyangdan@163.com;申德荣(1964—),女,博士,教授,主要研究方向为分布式数据库、Web数据管理等。

根据对问题领域的认识和 Web 服务请求集合、用户的限制条件等，模糊地或是精确地定义 Web 服务请求流程(或模板)。在合成阶段，如何根据用户的偏好和 Web 服务的 QoS 来寻找最优的 Web 服务进行合成，方法一是根据目标生成 Web 服务请求流程，选择 Web 服务提供商，生成可执行的合成服务实例；方法二是为每个 Web service 请求查找到具体的 Web 服务，得到一个 Web 服务集合，在这个集合上生成可执行的合成服务。方法二提供了动态选择和绑定 Web 服务进行合成的方法。在执行阶段，通过调用执行引擎执行合成服务流程，同时考虑到执行阶段的流程监控、事务处理、异常处理等问题。

2.2 Web 服务合成系统架构

本文基于 Web 服务合成的生命周期提出了一个 Web 服务合成的架构。根据生命周期进行架构的设计符合对合成服务管理的规律，使架构更加清晰，有利于原型系统的构建。此架构包括合成服务建模器、合成服务合成器、合成服务执行器 3 大部分，对 Web 服务合成提供了从定义、合成到执行各个阶段的支持。系统架构如图 1 所示。

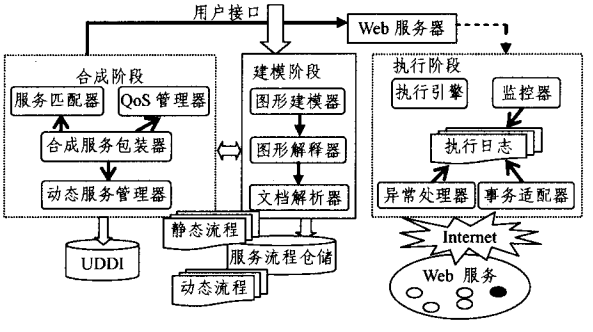

图 1 Web 服务合成架构

(1)建模阶段对应的建模器工具：提供对合成服务的定义支持和解析定义到任务仓和规则仓存储。包括一个图形化的流程建模器、图形解释器、合成服务定义语言的文档解析器。服务仓储存储解析了的合成服务流程。流程分为两种：静态流程(或模板)和动态流程(或模板)。

(2)合成阶段对应的合成工具：提供对服务的合成和部署、发布。包括合成 Web 服务包装器，将用户定义的合成服务包装成一个普通的 Web 服务，并实现对合成 Web 服务的发布和部署。动态服务管理器处理执行中的动态服务绑定请求，其功能包括了动态服务查找和最优选择，还负责动态服务的消息消歧转换和调用。QoS 管理器对 Web 服务的 QoS 进行管理、评分统计等。

(3)执行阶段对应的执行工具：主要涉及到合成服务的执行引擎、监控器、异常处理器、事务适配器。执行请求的合成服务实例，读取服务仓储中的操作，调用参与流程的 Web 服务节点。监控器用来监控流程的执行情况。异常处理器负责处理执行中出现的异常情况，如服务调用出错、动态绑定失败等。事务适配器提供对合成服务参与者间的事务协调与支持。

3 建模阶段

目前 Web 服务合成定义语言的规范有 XLANG[4]，WSFL[5]，BPML[6] 等，以及最新的 BPEL4WS[7]。BPEL4WS 定义语言较复杂并且不支持动态服务绑定，只支持静态绑定已有的服务。但是实际上 Web 服务的更新和每天新发布的 Web 服务数量巨大，因此需要一种支持动态合成的 Web 服务合成定义语言来迎合这种高度动态的环境。因此在 BPEL4WS 的基础上，对其进行了不失一般性和通用性的简化和扩展，并利用语义 Web 服务增强了动态服务支持，提出了服务合成的定义语言 e_SPDL[8]。e_SPDL 基于 XML，引入了服务容器的概念，在服务参与者〈participant〉标签中引入 service container 属性，通过执行引擎的调用支持动态服务调用。在建模阶段，为了提供友善的用户使用界面，使用可视化的服务合成流程建模器、图形解释器、合成服务定义语言的文档解析器，完成从合成服务定义到最后存储到服务仓储中。使用原有已被采用的规范进行修改和扩展的好处，是可以利用已经被其他研究者广泛熟悉的规范；可以继续使用其它的 Web 服务合成规范。

4 合成阶段

4.1 服务匹配器

服务匹配器利用 DAML 定义本体元数据集和概念关系集，应用本体解决 Web 服务合成过程中存在的异构问题[9]。其主要功能包括服务查找、服务异构解决、标准服务定义。众所周知，Internet 是一个充满异构的环境，而 Web 服务更是来自不同的服务提供者。在服务查找和服务匹配时也存在如何解决异构性的问题，如方法名异构、操作参数结构异构等，执行时需要进行适当的映射转换来实现对服务的优质查找和匹配。通过增加语义描述能力，可以更准确、有效地找到符合要求的服务。

4.2 动态服务管理器

针对在合成服务建模阶段的两种不同的流程：静态流程和动态流程，在合成阶段使用不同的解决方法进行合成。

(1)静态绑定对于静态流程，要调用的服务的绑定地址已经在流程中确定。

(2)动态绑定对于动态流程，要调用的服务并不具体绑定并且不是唯一，可以有多个相同功能的候选服务，只是给出该服务及其操作的描述信息。服务的绑定延后到流程执行时进行。动态服务管理器根据服务描述信息来寻找和绑定、调用合适的服务。

4.3 QoS 管理器

随着功能相似的 Web 服务数量的爆炸性增长，利用 QoS 属性来对 Web 服务进行筛选，是进行 Web 服务合成实用化的关键。对于动态绑定的服务，加入 QoS 和用户偏好，使之更加符合用户的个性化要求和个人偏好，即基于服务的 QoS 进行查找服务质量高的服务。目前对 Web 服务 QoS 的评价方法多数都是把 Web 服务进行事前量化后进行的，一般的做法是把相应的 QoS 指标量化后分配权值，并进行计算。例如 W_1(平均响应时间)=0.35，W_2(同行评价)=0.3，W_3(服务价格)=0.35，则综合评价得分为 $Total=\sum_{1}^{m}(W_i * M_i)$。其中，$W_i$ 为 M_i 各项的权重，i 的取值为 $1,2,3,\cdots,m$，$\sum_{1}^{m}W_i=1$。这种方法没有考虑到这些权重因子或是影响因子可以根据一定的规则进行修改和进化。所以本文提出使用基于多值背景值依赖的方法来获取有用的值依赖，调整权重因子。把 Web 服务的 QoS 的取值作为语言型的多值背景，如表 1 所列。

表1 Web服务的QoS背景值

Web service	平均响应时间	同行评价	价格	接受否
ws1	M	E	C	Yes
ws2	S	B	C	Yes
ws3	S	E	H	Yes
ws4	M	G	H	Yes
ws5	L	G	C	No
ws6	M	B	H	No
ws7	L	E	C	No
ws8	L	B	H	No

其中，平均响应时间的取值 L 代表服务的响应时间长，S 代表服务的响应时间短，M 代表服务的响应时间中等。同行评价的取值 E 代表评价很好，G 代表评价较好，B 代表评价不好。价格的取值 C 代表服务的价格低，H 代表服务的价格高。其中平均响应时间、同行评价、价格为条件属性，接受否为决策属性。通过计算背景中的值依赖(属性蕴含)可以得出有用的值依赖如下：

$S \rightarrow Yes; MC \rightarrow Yes; L \rightarrow No$

这说明响应时间短的服务是被用户接受的，响应时间中等但是价格便宜的服务也是被用户接受的，而价格高的服务是不被接受的。通过这些值依赖的分析结果，可以适当调整相应的权重因子。例如，平均响应时间的权重因子调整为0.4，同行评价的权重因子调整为0.2，服务价格的权重因子调整为0.4。

5 执行阶段

在合成服务的执行阶段，引擎从服务调用点开始调用服务流程。通常，对服务调用点的定义包括了服务类型、服务的绑定信息、服务的补偿操作等。当某个Web服务被调用时，服务请求被解析，服务实例被交给合成服务执行引擎执行。执行引擎根据服务仓储中的定义，按流程执行顺序通过Web服务器调用相关Web服务。当执行中遇到动态绑定的服务时，引擎发送请求到动态服务管理器，动态服务管理根据用户定义在UDDI注册仓中查找满足要求的Web服务，从中选择最优的调用，并把执行结果返回给引擎。合成服务执行中将可能遇到的异常报告给异常管理器，异常管理器负责执行相关的恢复和补偿操作。

合成服务在执行阶段，经过执行引擎调用。服务提供了两种类型的操作：请求-响应型和单向操作。系统按照如下规则生成操作：

(1)请求-响应型操作。对于合成服务定义的receive节点，如果具有对应的reply节点，则该节点对被包装成生成的Web服务的一个请求-响应型操作，表示该操作在调用时立即返回结果。

(2)请求型操作。对于合成服务定义的receive节点，如果没有对应的reply节点，该receive节点被包装成一个单向操作。这种操作经常是作为合成服务接受参与者的异步消息端口。

合成服务将流程中所有的异步消息接受节点receive作为Web服务的操作发布。用户通过这个Web服务来调用相应的合成服务，而合成服务的参与者调用这个Web服务中的操作来传递合成服务中的异步消息。一个合成服务流程操作以receive开始，以reply或invoke操作结束，中间是任意类型的服务调用点。当一个流程是以reply结束，表示合成服务可以期待得到同步的响应，否则表示流程的执行结果异步返回，此时需要调用结果接受者合适的消息接受操作。需要说明的是，并不是流程中所有的receive操作都有对应的reply操作，如流程中等待某个invoke的异步应答。

5.1 监控器

由于Web服务是无状态的，并且Web服务的操作间也是独立的，对于一个合成服务，同时会有多个实例运行，因此执行阶段有必要监控流程执行的情况。监控器可以查看各个合成服务实例的运行状态，还可以为系统级的故障提供恢复。监控器主要通过向执行日志中记录来完成监控任务。执行日志负责维护合成服务实例的历史数据信息和执行状态；保证合成服务实例执行的连续性和一致性；及时发现和更正用户的误操作；执行事后审计、执行变化分析等。以旅游合成服务的订票、租车Web服务为例，执行日志文件如下所示：

```
〈? xml version="1.0" encoding="utf-8" ?〉
〈log〉
 〈process name="travelService" InstanceNo="0012" 〉
 〈DateTime〉2009-1-2 8:07:30〈/DateTime〉
 〈User name="Mike" /〉
 〈operation name="ticketBooking" type="invoke" nextOp="carRenting"〉
 〈startTime〉 2009-1-2 8:07:45〈/startTime〉
 〈endTime〉 2009-1-2 8:08:15〈/endTime〉
 〈MessageContainer InstanceNo="0005"〉
  〈input params=""/〉
  〈output params=""/〉
 〈/MessageContainer〉
 〈TraceLevel type="info"〉
 〈message〉the operation is executed successful〈/message〉
  〈/TraceLevel〉
  〈exception〉〈/exception〉
 〈/operation〉
 〈operation name="carRenting" type="receive" nexOp="carRentingReply"〉
  〈startTime〉 2009-1-2 8:08:45〈/startTime〉
  〈endTime〉 2009-1-2 8:09:10〈/endTime〉
  〈MessageContainer InstanceNo="0005"〉
   〈input params=""/〉
   〈output params=""/〉
  〈/MessageContainer〉
 〈TraceLevel type="warning"〉
 〈message〉the operation is executed successful but with warning.〈/message〉
  〈/TraceLevel〉
  〈exception〉〈/exception〉
 〈/operation〉
 〈operation name="carRentingReply" type="reply"〉
  〈startTime〉 2009-1-2 8:09:45〈/startTime〉
  〈endTime〉 2009-1-2 8:10:05〈/endTime〉
  〈MessageContainer InstanceNo="0005"〉
   〈input params=""/〉
   〈output params=""/〉
  〈/MessageContainer〉
  〈TraceLevel type="error"〉
  〈message〉the operation is failed.〈/message〉
  〈/TraceLevel〉
   〈exception〉〈/exception〉
 〈/operation〉
  〈/process〉
  〈process〉
 ……
  〈/process〉
 ……
```

〈/log〉

日志中包含了下步操作节点信息，但系统实际并不依据这个信息来调用下步操作节点，这个信息更多的作用是作为验证，因为系统是根据合成服务的定义生成的合成 Web 服务操作。从合成服务执行的角度看，合成 Web 服务操作是合成服务执行的各个阶段的异步消息入口点。操作本身包含了在合成服务执行中的位置信息，系统能够根据被调用的 Web 服务操作确定合成服务内部的下一步动作。

5.2 基于执行文件的异常处理器和事务适配器

系统中提供了执行异常处理机制，依靠执行阶段的日志进行异常处理。当执行出现异常时，首先由执行监控器写入执行日志，异常处理器解析日志中 XML 文档片段〈exception〉〈/exception〉的部分进行处理。在一个 Web 服务环境下对事务的参与者没有公共的事务语义、事务上下文表达和协调协议。在一个合成服务中，参与者来自不同的服务提供者，可能来自异构的平台，还有各自的商业规则，而且是完全自治的独立实体。传统的事务的 ACID 特性不能完全适用于 Web 服务，传统的事务模型不能完全适用于 Web 服务合成。分布的 Web 服务的事务管理要求 Web 服务的事务支持和协调。引入事务适配器[10]的目的是为了把合成逻辑和事务补偿逻辑相分离，把流程中的事务要求封装在事务适配器中。本系统中事务适配器依靠执行日志中的 XML 文档片段〈transaction〉〈/transaction〉的部分，根据流程中定义执行相应事务处理，进行补偿或恢复操作。

结束语 本文基于 Web 服务合成生命周期的定义、合成、执行 3 个阶段，提出了较为完善、有效的 Web 服务合成系统架构，综合考虑了 Web 服务合成的各个方面以及各个阶段所涉及到的问题，如合成定义语言、动态服务绑定、服务的异构解决、流程执行监控、异常处理、事务处理等，为 Web 服务合成的进一步研究提供了较好的参考架构。

参 考 文 献

[1] Casati F, Krishnamoorthy S I. Adaptive and Dynamic Service Composition in eFlow[EB/OL]. Software Technology Laboratory HP Laboratories

[2] Sheng Q Z, Benatallah B, Dumas M. SELF-SERV: A Platform for Rapid Composition of Web Services in a Peer-to-Peer Environment [EB/OL]. University of New South Wales Marlon Dumas Queensland University of Technology

[3] Pires P F, Benevides R F M, Mattoso M. WebTransact: A Framework for Specifying and Coordinating Reliable Web Service Compositions[EB/OL]. http://www.cos.ufrj.br/~pires/WebTransact.html. 2002

[4] Thatte S. XLANG: Web Services for Business Process Design [EB/OL]. http://www.gotdot.net.com/team/xml_wsspecs/xlang-c/default.htm

[5] Leymann F. WSFL: Web Service Flow Language 1.0[EB/OL]. http://www-3.ibm.com/software/solutions/Webservices/pdf/WSFL.pdf

[6] Arkin A. Business Process Modeling Language [EB/OL]. http://www.bpmi.org/BPML

[7] Curbera F, Goland Y, Klei J, et al. BPEL4WS: Business Process Execution Language for Web Services Version1.0 [EB/OL]. http://www-900.ibm.com/developerEorks/cn/Webservices/ws-bpel_spec/index.shtml

[8] 杨丹，申德荣，等. 一种支持 Web 服务合成的模型定义语言-e_SPDL[J]. 计算机集成制造系统-CIMS，2003，9(10)：932-936

[9] 张蓉，申德荣，等. 基于本体的 Web 服务查找和合成技术研究[J]. 计算机集成制造系统-CIMS，2003，9(10)：921-925

[10] Schafer M, Nejdl W. An Environment for Flexible Advanced Compensation of Web Service Transactions[J]. ACM Transactions on the Web，2008，2(2)

(上接第 127 页)

将原本不具备监控能力的软件改造为可被监控的软件系统，使得运行时可以监视软件的内部运行状态，从而提高其可靠性和安全性。

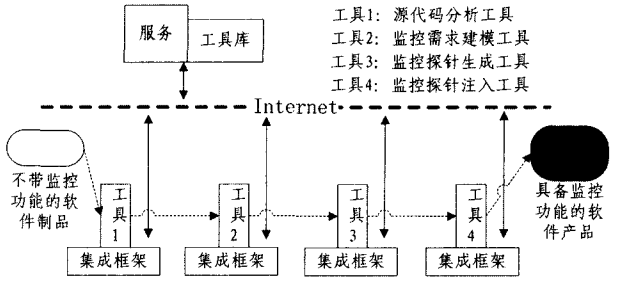

图 6 分布式监控软件生产线示意图

图 6 为分布式监控软件生产线的示意图，该生产线的功能由分布在 Internet 上的 4 个工具来实现，并由服务器进行统一的协同管理，有效地实现了分布式、协同化的监控软件生产。

结束语 本文提出了一种 Internet 环境下的分布式软件生产线框架，详细阐述了该框架的体系结构，给出了基于此框架创建、运行分布式软件生产线的方法和详细流程，并介绍了基于此框架开发的一个分布式监控软件生产线。实践证明，本框架能够有效支持 Internet 环境下分布式、协同化的软件生产，具有一定的理论及应用价值。

参 考 文 献

[1] Tracz W. Confessions of a Used Program Salesman-Institutionalizing Software Reuse[M]. New York, NY: Addison-Wesley Publishing Co., April 1995

[2] Yang F Q, Mei H, Li K Q. Software reuse and software component technology[J]. Acta Electronica Sinica, 1999, 27(2): 68-75

[3] Szyperski C. Component Software: Beyond Object-Oriented Programming(2nd ed)[M]. Boston: Addison-Wesley Professional, 2002

[4] Veryard R. Component-based business, plug and play[M]. London: Springer, 2001

[5] Yang F Q. Thinking on the development of software engineering technology[J]. Journal of Software, 2005, 16(1): 1-7

[6] Northrop L M, Clements P C, Bachmann F, et al. A Framework for Software Product Line Practice[EB/OL]. Version 5.0. 2007. http://www.sei.cmu.edu/productlines/framework.html

[7] Yang F Q, Mei H, Li K Q, et al. The summary of JB III supporting components reuse[J]. Computer Science, 1999, 26(5): 50-55

[8] http://www.osgi.org/About/Technology

[9] http://www.eclipse.org

[10] Kiczales G, Lamping J, Mendhekar A, et al. Aspect-Oriented Programming[C]// Proceedings of the European Conference on Object-Oriented Programming. 1997, 1241: 220-242

[11] A look at aspect-oriented programming[OL]. http://www.ibm.com/developerworks/rational/library/2782.html