VxBPELEngine:一种变化驱动的适应性服务组装引擎

孙昌爱 薛铁恒 胡长军

(北京科技大学计算机与通信工程学院 北京 100083)

摘 要 近年来,面向服务的架构(SOA)正逐渐成为分布式系统开发的新范型.为了满足快速变化的需求,服务组装应具备足够的适应性.针对目前广泛采纳的服务组装语言BPEL在适应性支持方面存在的不足,对标准BPEL进行扩展,开发了VxBPEL,支持服务组装中的可变性设计.为了在运行时刻解释与执行服务组装中的可变性定义与可变性配置,基于开源BPEL引擎ActiveBPEL开发了VxBPEL引擎VxBPELEngine.通过实例系统验证了基于可变性设计的适应性服务组装方法的可行性,评估了VxBPELEngine引擎的性能.

关键词 Web服务;BPEL;VxBPEL;可变性管理;服务组装引擎

中图法分类号 TP311 **DOI号** 10.3724/SP.J.1016.2013.02441

VxBPELEngine: A Change-Driven Adaptive Service Composition Engine

SUN Chang-Ai XUE Tie-Heng HU Chang-Jun

(*School of Computer and Communication Engineering, University of Science and Technology Beijing, Beijing* 100083)

Abstract Service-Oriented Architecture (SOA) has evolved as a mainstream development paradigm for distributed systems in recent years. To embrace quickly changing requirements, service compositions are expected to be adaptive. However, BPEL, a widely recognized standard service composition language, is limited in support for adaptability within service compositions. In order to overcome this limitation, we developed the VxBPEL in our previous work, which is an extension of BPEL to enable the variability design during service compositions. To interpret the variability definition and configuration of service compositions at runtime, we implemented the VxBPEL engine called VxBPELEngine, by reusing and extending the ActiveBPEL, an open-source BPEL engine. Two case studies have been conducted to validate the feasibility of the variability design-based adaptive service composition approach, and evaluate the performance of the VxBPELEngine.

Keywords Web services; BPEL; VxBPEL; variability management; service composition engine

1 引 言

近年来,面向服务的架构(SOA)逐渐成为Internet环境下的应用程序开发的一种主流范型或方法学[1].由于单个服务往往无法满足实际需求,需要将多个服务按照一定的方式协调组织起来以支持复杂的业务过程,这样的过程称为服务组装(也称服务组合).与传统应用程序开发范型相比,基于SOA的软件开发呈现出若干新特点[2].这些特点使得基于SOA

收稿日期:2011-12-27;最终修改稿收到日期:2012-09-01.本课题得到国家自然科学基金(60903003,61370061)、北京市自然科学基金(4112037)、国家"十二五"科技支撑计划(2011BAK08B04)、中央高校基本科研业务费资助项目(FRF-SD-12-015A)、北京市优秀人才培养资助项目(D类)(2012D0090060000002)资助.孙昌爱,男,1974年生,博士,副教授,中国计算机学会(CCF)高级会员,主要研究方向为软件测试、软件体系结构、服务计算.E-mail:casun@ustb.edu.cn.薛铁恒,男,1987年生,硕士研究生,主要研究方向为软件工程、服务计算.胡长军,男,1963年生,博士,教授,博士生导师,主要研究领域为软件工程、高性能计算.

的软件开发可以较好地解决分布、动态、异构环境下,数据、应用和系统集成等问题,支持快速的业务重整与优化.围绕 Web 服务及其服务组装,开发了一系列的标准、语言与支持工具[3].

BPEL[4]是一个遵循 SOA 基本原理,支持面向过程的可执行服务组装语言,在学术界和工业界受到广泛重视.基于 SOA 的系统通常部署和运行于一个动态、开放的环境中,实现的业务过程本身往往是快速变化的,面向差别极大的用户群.这意味着,采用服务组装实现的 SOA 系统,应具备足够的灵活性,能够适应快速变化的需求.围绕着适应性服务组装的相关研究工作近年来非常活跃[5].特别地,在基于服务组装语言 BPEL 实现适应性服务组装方法方面,已有工作探讨了如何基于 Aspect 技术或采用代理机制解决服务组装的适应性问题[6-9].提出的适应性服务组装方法,都是面向运行时的服务组装实例,仅从服务组装的实现层考虑适应性问题.这些方法存在不易维护、技术实现复杂、影响了服务组装性能等缺点.

前期工作中,我们探讨了如何从服务组装的规格说明层考虑适应性问题,提出一种基于可变性设计的适应性服务组装方法[10-13].该方法在服务组装规格说明中引入可变性设计,在部署和运行阶段支持可变性设计与配置的解释执行;通过增强服务组装在设计、部署和运行时的可变性,显式地表达和管理服务组装中不稳定或可变的因素,即将可变因素作为第一类设计对象并实施可变性管理.提出的方法较好地解决了已有适应性服务组装方法中存在的不易维护、技术实现复杂、影响了服务组装性能等问题.

本文讨论 VxBPEL 引擎 VxBPELEngine 的设计与实现关键问题,采用实例研究的方式验证所提适应性组装方法的可行性,评估 VxBPELEngine 性能.主要贡献包括:

(1)一种适应性服务组装引擎 VxBPELEngine:通过复用和扩展 ActiveBPEL 引擎,实现 VxBPEL 引擎 VxBPELEngine 的开发;VxBPELEngine 支持多个 BPEL 版本(包括 BPEL4WS 1.1 与 WS-BPEL 2.0).

(2)采用 2 个服务组装实例系统,验证了基于可变性设计的适应服务组装方法的可行性,评估了 VxBPELEngine 的性能.

2 背景介绍

本节介绍 BPEL 与 VxBPEL 的基本原理与相关概念.

2.1 BPEL

SOA 定义了 Internet 环境下松散耦合的、基于标准的、面向服务的应用程序开发模式[2].在该开发模式中,服务提供者开发并拥有服务,按照一定的标准(如 WSDL[3])对服务进行描述,通过 UDDI[3]将服务发布到服务代理;服务代理包含一个可用服务的存储库,允许感兴趣的服务使用者查找相关的服务;服务使用者通过对服务代理的查询,将查询到的服务进行绑定,根据服务描述的接口调用服务功能.SOA 软件的基本单元为服务,通过对外提供一组操作参与分布式计算,实现一个软件系统或软件系统的一部分.Web 服务是 SOA 基于 Web 的实现,由于单个 Web 服务往往无法满足实际需求,需要将多个 Web 服务协调与组织起来支持某个业务过程,上述过程称为服务组装.

BPEL 是一个支持面向过程、可执行的 Web 服务组装语言.BPEL 过程包括伙伴链接(partnerLink)声明、变量声明、处理器(handler)声明以及业务过程描述 4 个部分.BPEL 描述的业务过程是一组活动以及活动之间的控制流.活动分为基本活动与结构型活动.其中,invoke 是最常使用的基本活动类型,刻画了业务过程与外部 Web 服务之间的交互.结构型活动由基本活动与结构型活动组成,其描述包含了顺序、分支、循环等常见的控制逻辑以及条件判断和并行机制.标准版本的 BPEL 在描述动态、多变的业务过程时存在明显不足[12].

2.2 VxBPEL

可变性是软件系统能够在特定环境中扩展、改变、定制或配置的能力[10].可变性的核心概念包括变异点(Variation Point)、变体(Variant)和依赖关系(Dependency).变异点是软件系统中可能存在变化的部分.通常,一个变异点存在多个设计方案可供选择,其中每一个选择方案称为变体.当每个变异点都选择一个变体后,选择的变体集合被称作可变性配置.依赖关系规定不同变异点对应的变体之间的约束关系.可变性管理包括设计、使用和维护可变性.

实现适应性服务组装的关键在于如何识别、表达与执行服务组装中潜在的各种变化,即将变化作为第一类设计对象进行处理.可变性管理最初是为了解决软件产品线领域中多个软件产品之间的复用问题[14].借鉴可变性管理框架 COVAMOF[15],我们开发了面向服务组装的可变性管理框架,如图 1 所示:

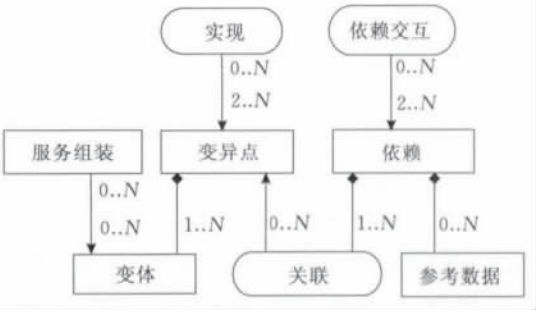

图 1　面向服务组装的可变性管理框架元模型

(1) 变异点. 代表提供选择的位置. 变异点有很多特征，如变化类型、抽象层次、绑定时间和基本原理.

(2) 变体. 代表变异点中可行的选项.

(3) 实现. 由于变异点可存在于不同的抽象层次中，实现关系指明了如何通过低层抽象中选择变体，以实现高层抽象中变异点的选择的规则.

(4) 依赖. 代表了一种系统属性，并且说明变异点的绑定如何影响这个属性值，即选择特定的变体如何影响该属性的值. 依赖包含关联和参考数据，细化为依赖交互.

(5) 依赖交互. 描述依赖关系的一个条目.

(6) 关联. 指影响系统属性值的变异点. 对于与某个依赖相关的每个变异点，关联关系是依赖关系的一部分，因此，关联定义了与变异点的某种联系.

(7) 参考数据. 定义了诸如变异点绑定和相应的系统属性值等实体的信息. 这些系统属性的值是通过测试获得的.

依据上述可变性管理框架的元模型，我们开发了 VxBPEL[10]. VxBPEL 中的可变性构造子(即实现可变性管理中的各种概念)采用 XML 表示，与现有服务组装语言 BPEL 表示法一致. 这样设计方案有利于继承 SOA 在解决应用程序集成和数据集成方面的突出优点.

变体是可变性设计中的核心概念之一，其定义方式如图 2 所示. 其中，〈VXSpace：Variant〉标识了变体，“VXSpace”是定义了 Variant 类型定义的名字空间. 变体通过名字 name 进行引用和标识.

```
〈VXSpace:Variant name= $variant_name〉
    〈 !-original BPEL activity--〉
〈/VXSpace:Variant〉
```

图 2　VxBPEL 中变体定义的语法格式

变异点是可变性管理另一核心概念，定义方式如图 3 所示. 变异点由名字(name)和一组变体(Variants)组成. 其中，〈VXSpace：VariationPoint〉标识了变异点，“VXSpace”是定义了 VariationPoint 类型定义的名字空间. 变异点通过名字 name 进行引用和标识. 实现该变异点的变体集合由〈VXSpace：Variants〉标识，包含的变体由〈VXSpace：Variant〉进一步定义.

```
〈VXSpace:VariationPoint name= $VariationPoint_name〉
   〈VXSpace:Variants〉
      〈 !-define variants here--〉
   〈/VXSpace:Variants〉
〈/VXSpace:VariationPoint〉
```

图 3　VxBPEL 中变异点定义的语法格式

复杂的 BPEL 流程中可能包含多个变异点，这些变异点之间存在依赖关系. 采用可变性配置来描述这些可变性元素之间的依赖关系. 可变性配置的定义方式如图 4 所示. 可变性配置由名字(Name)、唯一标识号(id)、初始化缺省变体(defaultVariant)、基本原理(Rationale)、一组变体(Variants)组成. 其中，可变性配置由〈VXSpace：ConfigurableVariationPoint〉标识，ID 属性唯一标识可变性配置，通过

```
〈VXSpace:ConfigurableVariationPoint id= $ID defaultVariant=default_variant_name〉
    〈VXSpace:name〉…… 〈/VXSpace:name〉
    〈VXSpace:Rationale〉……〈/VXSpace:Rationale〉
    〈VXSpace:Variants〉
      〈VXSpace:Variant name= $variant_name〉
        〈VXSpace:VariantInfo〉…… 〈/VXSpace:VariantInfo〉
        〈VXSpace:RequiredConfiguration〉
          〈VXSpace:VPChoices〉
            〈VXSpace:VPChoice vpname= $vpname variant= $variant_name/〉
          〈/VXSpace:VPChoices〉
        〈/VXSpace:RequiredConfiguration〉
      〈/VXSpace:Variant〉
        〈 !—define other variants here--〉
    〈/VXSpace:Variants〉
  〈/VXSpace:ConfigurableVariationPoint〉
```

图 4　VxBPEL 中可变性配置定义的语法格式

defaultVariant 属性可指定缺省的实现变体.〈VXSpace:Rationale〉和〈VXSpace:VariantInfo〉分别说明了可配置变异体和变体的相关信息.每个变体所需要的配置信息由〈VXSpace: RequiredConfiguration〉标识.可变性配置细化为一组变体的选择,由〈VXSpace:VPChoices〉标识.对每个变体选择的说明由〈VXSpace:VPChoice 〉标识,vpname 属性指明了该选择的名字,variant 属性通过名字引用已定义的变体.通过〈VXSpace:RequiredConfiguration〉和〈VXSpace:VPChoices〉的联合使用,可以实现复杂的可变性配置(指明各种变体之间复杂的实现与依赖关系).

目前,VxBPEL 支持服务组装中的可变性设计类型包括:(1)具有相同接口的服务替换;(2)具有不同接口的服务替换;(3)服务调用时参数的改变;(4)服务组装或部分活动的重组.

3 VxBPELEngine 的设计与实现

标准 BPEL 引擎无法解释执行包含可变性设计的服务组装规格说明.通过对标准 BPEL 引擎 ActiveBPEL[16] 的复用与扩展,我们开发了一个支持 VxBPEL 引擎 VxBPELEngine.下面详细讨论 VxBPELEngine 设计与实现的关键问题.

3.1 VxBPELEngine 的基本原理

基于可变性设计的适应性服务组装方法的基本原理如图 5 所示.首先,采用标准的 BPEL 语言描述基本的业务过程,得到基本的服务组装规格说明(Basic Service Compositions);然后,采用 VxBPEL 中的可变性构造子对基本的服务组装规格说明实施可变性设计,得到适应性服务组装规格说明(Adaptive Service Compositions).采用该方法得到的服务组装规格说明中,不仅包含标准的 BPEL 元素,也包含 VxBPEL 设计元素.在运行时刻,适应性服务组装引擎 VxBPELEngine 根据系统可变性配置,执行不同的业务过程.

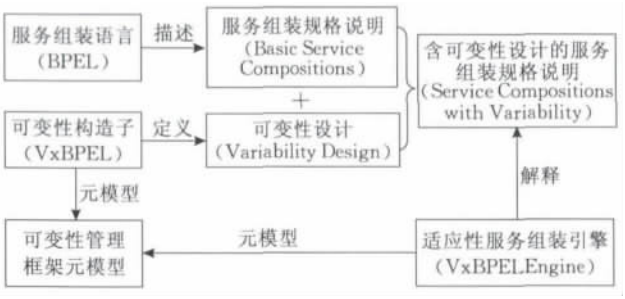

图 5 基于可变性设计的适应性服务组装方法

为了解释与执行服务组装规格中可变性设计构造子,必须对已有 BPEL 引擎进行扩展.图 6 示意了 VxBPELEngine 解释执行含可变性设计的服务组装规格说明的基本原理.这里我们采用了"统一存储,分别解释"的思想解释与执行 VxPBEL 元素.其中,BPEL 解释器负责解释标准 BPEL 元素;VxBPEL 解释器负责解释 VxBPEL 元素.VxBPELEngine 是一个解释型编译系统,VxBPEL 解释器的设计与实现遵循了图 1 所示的可变性管理框架元模型.

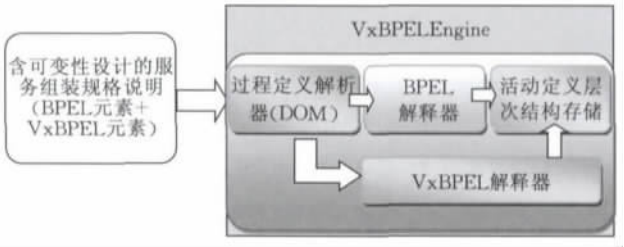

图 6 VxBPEL 解析的基本原理

首先,VxBPELEngine 构造活动定义层次结构图,解析构成过程的活动以及活动之间的依赖关系.过程定义解析器读取过程定义(即 VxBPEL 描述的服务组装规格说明),并负责解析任务的分发.具体说来,如果当前读取的过程定义元素属于标准的 BPEL 元素,则分发给标准的 BPEL 解释器;如果属于 VxBPEL 定义的可变性构造子元素,则分发给 VxBPEL 解释器.标准的 BPEL 解释器和 VxBPEL 解释器依据活动定义创建相应的活动对象.过程定义解析完成后,VxBPELEngine 将存储过程的活动定义层次结构,并依此进行随后的过程部署、执行以及运行时重新配置.

3.2 VxBPELEngine 的设计

目前存在多个版本的 BPEL 引擎,具有代表性的包括 ActiveBPEL、Apache ODE、IBMWebSphere Process Server、Microsoft BizTalk Server、JBoss jBPM 等.其中,ActiveBPEL 是一个支持 BPEL 规范的开源流程引擎,具有良好的结构和实现方式.我们选择基于 ActiveBPEL 引擎设计与实现 VxBPELEngine,在设计 VxBPELEngine 时,遵循的设计原则是在不改变标准 BPEL 引擎结构的前提下,最大限度保留和复用 ActiveBPEL 引擎机制.图 7 示意了 VxBPELEngine 的架构.

(1)右侧部分包含了用于存储过程的相关文件和信息.Deployment Plans(部署计划)保存过程部署描述文件(.pdd)、业务过程执行文件(.bpel)、Web

服务描述文件(.wsdl)以及 Partner 文件；Process State(过程状态)记录过程的状态，如所需消息是否到来、当前活动是否执行完毕等；Queues(队列)和 Alarms(警报)用于消息的管理.

(2)核心部分是 BPEL Processor. 该模块通过 Process Creation&Management(采用易于扩展的工厂设计模式实现，能够支持不同版本的 BPEL 过程)管理引擎的创建、调用 Queues and Alarms(队列与警报管理服务)和 Timer Service(计时服务)管理相应的服务，采用过程配置使得引擎的各项职责互相分离. BPEL Processor 首先从 Deployment Plans 中提取 BPEL 文件，传递给 Process Creation & Management 单元，后者依据 BPEL 树形结构创建数据结构，配置相应属性. 通过读取 aeEngineConfig.xml 文件实现引擎的配置处理. 该文件描述了缓存的大小和日志状态，决定了队列和警报管理服务、流程管理器(Manager)、部署日志、部署处理器(Deployment Handler)等启动位置.

(3)一些辅助模块包括：Admin and Event Handling 模块用于引擎、事件等管理，主要任务是管理和控制引擎、确保部署的流程正常执行等；Partner Addressing 模块为业务流程中交互的每个伙伴(partner)提供寻址信息(即 Endpoint reference). 寻址信息存储在部署描述符、消息上下文或者伙伴定义对象中；Manager 模块管理各种存储结构，包括流程管理者(用于管理流程的，如获取流程 ID、创建业务流程等)和队列管理者(用于管理到来的数据和已经被执行却正在等待数据的活动).

(4)在 Process Creation&Management 单元中加入 VxBPEL Management 模块，用于处理变异点和变体等 VxBPEL 元素. 由于每个变异点如何选择变体取决于 VxBPEL 文件中的可变性配置，为此，增加一个 Configuration 模块. 该模块负责读取与分析 VxBPEL 文件中的可变性配置信息. VxBPEL Management 依据可变性配置信息选择变体，创建相应的过程. 创建后的 BPEL 过程是标准的 BPEL 过程(VxBPEL 元素已被消解). 在运行时刻，通过 Configuration 模块更改可变性配置实现运行时可配置.

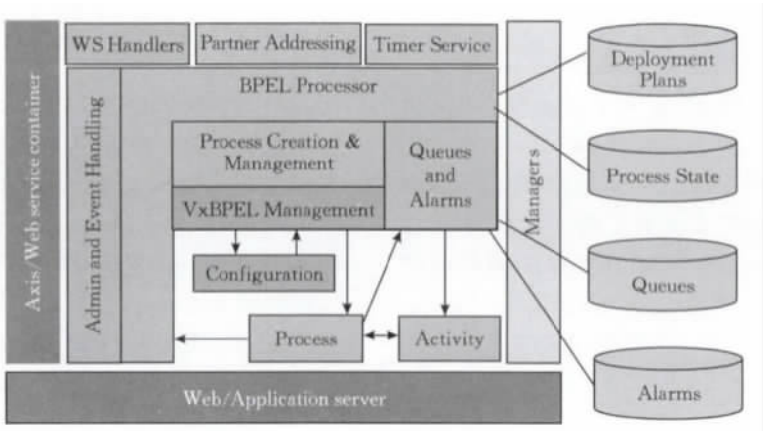

图 7　VxBPELEngine 架构

VxBPELEngine 按照特定的顺序执行过程中的各个活动，具体的运行过程如图 8 所示. 首先，BPEL Processor 根据 VxBPEL 活动定义(Deployment Plans)，开始进行过程实例 Process 的创建(步骤 1 和 2). 活动定义(包含活动之间的层次关系)以数据流形式传给 Process Creation&Management(步骤 3)，BPEL 活动依据其活动定义创建活动执行对象，VxBPEL 活动通过查询可变性配置信息 Configuration(步骤 4 和 5)，选取执行方案，调用 VxBPEL Management 完成含可变性过程的活动执行对象的生成(步骤 6). 已创建的 VxBPEL 活动执行对象结合已有 BPEL 活动执行对象，完成对 Process 的创建(步骤 7). Process 由一组活动执行对象 Activity 组成，Process 经过分析分解为特定顺序的活动队列(步骤 8)，生成的活动队列存储在 Queues 和 Alarms 中(步骤 9). 根据过程状态(Process State)(如要判断执行活动的所需消息是否到来、该活动是否执行完毕等)(步骤 10 和 11)，选择 BPEL Processor 执行活动的下一步动作(步骤 11). 其中，Alarms 存储 pick 等活动，该类活动需要等待某个具体事件发生(如基于计时器的警报信号)，从而处理所指定的活动.

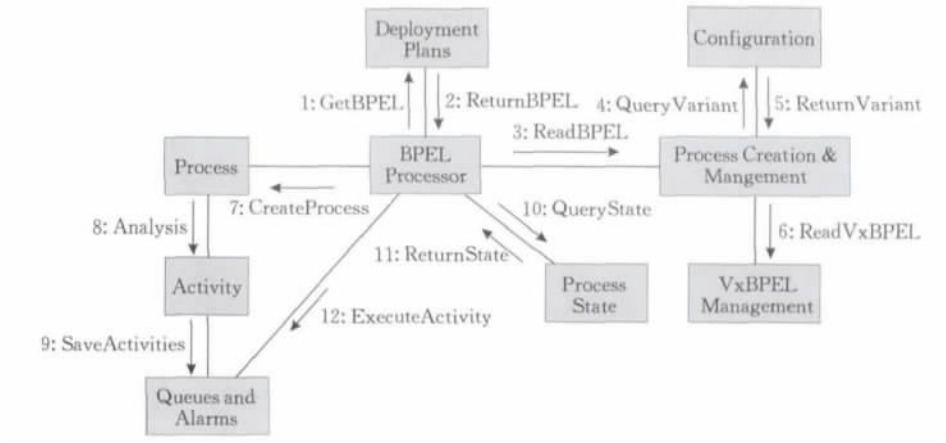

图 8 VxBPELEninge 的运行过程

3.3 VxBPELEngine 实现的关键问题

为了不改变 ActiveBPEL 引擎的结构，增加了一个可变性处理类（VarConfInfo），记录 VxBPEL 过程定义中的可变性设计与配置信息. 通过修改 ActiveBPEL 的配置文件 aeEngineConfig.xml，实现对相关可变性元素的管理. 可变性元素的识别可以通过名字空间进行区分. 例如，〈VXSpace：Variant〉中“VXSpace”给出了“Variant”类型定义的名字空间.

（1）VxBPEL 元素的解释与部署

当 ActiveBPEL 引擎读入一个活动后，调用过程模型的活动定义（Activity Definition），根据过程定义为其创建一个对象. 活动定义包含了 BPEL 活动的执行对象（即活动定义类的实例化对象）所需的相关信息. 引擎和事件监听模块在必要的时候访问相应的活动定义. 为了能够处理 VxBPEL 元素，必须扩展相关的活动定义、创建阅读器（Reader）. VxBPEL 解释器依据该 VxBPEL 元素的活动定义模型创建相应的执行对象. 处理 VxBPEL 变异点时，根据可变性配置信息在一次执行过程中选择一个变体. 在扩展 VxBPEL 元素的活动定义与阅读器时，遵循了 ActiveBPEL 的设计思想，即采用访问者（Visitor）设计模式为 VxBPEL 元素定义活动定义与创建执行对象，实现对 ActiveBPEL 引擎的无缝扩展.

（2）VxBPEL 元素的正确性验证

在执行 VxBPEL 规格说明之前，VxBPELEngine 引擎首先验证 VxBPEL 过程定义是否正确、是否遵循规范. ActiveBPEL 引擎中提供了对标准 BPEL 过程定义的检测机制，包括对结构化活动首个子元素的检测、基本活动语法是否符合规格说明、是否创建实例检测等. VxBPELEngine 复用了 ActiveBPEL 引擎检验机制，增加了对过程定义中可变性设计与配置的定义是否遵循 VxBPEL 规范的检测，修改了对结构化活动的子元素的检测机制. 例如，〈VariationPoint〉能够存在于〈flow〉、〈sequence〉等结构化活动中，通过扩展检测结构化活动子元素的检测机制，实现对 VxBPEL 过程定义的正确性进行验证.

（3）运行时可变性配置实现

实现运行时可变性配置的关键之处在于如何读取与部署 VxBPEL 元素，其原理如图 9 所示. 首先，

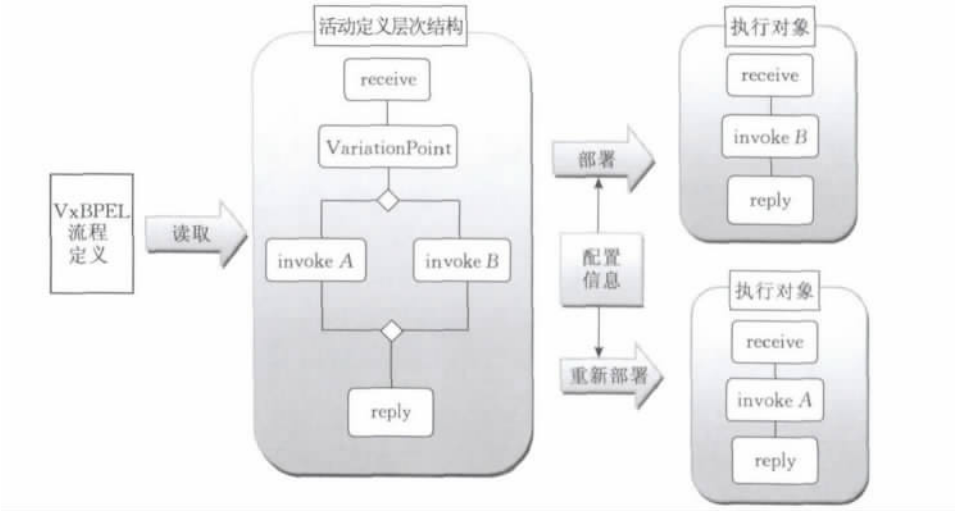

图 9 VxBPEL 过程定义的读取与部署原理图

读取 VxBPEL 过程定义，按照 VxBPEL 规格说明的层次关系创建过程实例对象，即活动定义层次结构(图中间部分). 初次读取与部署过程时，可变性处理类(VarConfInfo)记录了过程实例的过程定义和部署信息. 部署时，VxBPEL 解释器通过解析可变性配置，将 VxBPEL 扩展部分的活动定义转换为标准 BPEL 过程的执行对象形式(图右边部分).

为了支持用户需求的改变，可以借助 MX4J 或者 JConsole 技术修改 VxBPEL 过程的可变性配置. 依据新的配置方案，重新部署 VxBPEL 过程定义. 这里提出的适应性方法，从服务组装规格说明层次上解决适应性问题，具有“一次设计、多次运行”等优点.

4　实例研究

通过对 ActiveBPEL5.0.2 的复用与扩展，VxBPELEngine支持BPEL4WS 1.1与WS-BPEL 2.0. VxBPELEngine 可以运行在任何 Java Servlet 容器中，包含 102270 行 Java 代码(其中 VxBPEL 解释器的代码行数为 1224). 下面，我们采用两个实例系统示例与验证基于可变性设计的适应性服务组装方法的可行性，对 VxBPELEngine 的时间性能与可扩展性进行评估.

4.1　贷款核准 LoanApproval

在贷款核准场景中[4]，用户通过一个服务端口发送贷款请求(包括个人信息与贷款数量). 依据贷款请求信息，贷款核准服务执行业务审批流程后，返回“贷款成功”或者“贷款失败”. 影响贷款请求是否通过的因素，包括贷款数量与客户的风险值. 对于贷款数量小(低于 10 000)或低风险客户，贷款核准自动通过贷款请求；对于贷款数量较大且高风险客户，贷款核准程序相对复杂. 因此，处理贷款请求时，贷款核准服务首先调用“风险评估”服务快速评估客户风险值；对于无法自动通过核准的贷款请求，需要启动全面的核准程序，调用“贷款评审”服务对贷款请求进行深度评估.

采用 Web 服务组装实现上述贷款核准服务时，牵涉到客户一些重要个人信息. 该服务应该提供不同的访问模式，例如，对于普通的贷款请求，提供默认访问模式；对于大额度的贷款请求，提供加密模式. 采用 VxBPEL 实现上述服务组装时，可将访问模式定义为变异点，使用〈vxbpel：VariationPoint〉进行封装. 两个变体分别为默认模式与加密模式，使用〈vxbpel：Variant〉定义，变体名称分别为“default”和“encryption”. 图 10 示意了含可变性设计的贷款核准服务的服务组装规格说明. 通过 JConsole(图 11)修改可变性配置(图 12)，在运行时刻 VxBPELEngine 执行不同的 BPEL 过程场景. 图 13 和图 14 分别示意了默认访问模式和加密模式下的执行场景.

```
〈vxbpel:VariationPoint name=" receive"〉
  〈vxbpel:Variants〉
    〈vxbpel:Variant name=" default"〉
      〈receive createInstance="yes" name=" receive1" operation=" approve"
       partnerLink=" customer" portType=" apns:loanApprovalPT" variable=" request"〉
        ……
      〈/receive〉
    〈/vxbpel:Variant〉
    〈vxbpel:Variant name=" encryption"〉
      〈sequence〉
        〈receive createInstance="yes" name=" receiveEnc" operation=" approve"
         partnerLink=" customer" portType=" apns:loanApprovalPT" variable=" request"〉
         〈/receive〉
          ……
        〈invoke inputVariable=" decryptRequest" name=" decryptReceive"
         operation=" decrypt" outputVariable=" decryptResponse"
         partnerLink=" encryptionServiceLinkType" portType=" enc:EncryptionWebService"〉
        〈/invoke〉
          ……
        〈/sequence〉
      〈/vxbpel:Variant〉
    〈/vxbpel:Variants〉
〈/vxbpel:VariationPoint〉
```

图 10　贷款核准服务中基于 VxBPEL 的可变性设计

图 11 借助 JConsole 修改运行时可变性配置示意图

```
〈vxbpel:VariabilityConfigurationInformation〉
 〈vxbpel:ConfigurableVariationPoints〉
   〈vxbpel:ConfigurableVariationPoint id=" encryption" defaultVariant=" unencrypted"〉
     〈vxbpel:Name〉
          Encryption scheme
        〈/vxbpel:Name〉
     〈vxbpel:Rationale〉
          It is possible to configure the loan approval process to support encryption.
        〈/vxbpel:Rationale〉
     〈vxbpel:Variants〉
      〈vxbpel:Variant name=" unencrypted"〉
        〈vxbpel:VariantInfo〉
          Information is received unencrypted.
        〈/vxbpel:VariantInfo〉
        〈vxbpel:RequiredConfiguration〉
          〈vxbpel:VPChoices〉
            〈vxbpel:VPChoice vpname=" receive" variant=" default"/〉
            〈vxbpel:VPChoice vpname=" approver" variant=" default"/〉
            〈vxbpel:VPChoice vpname=" assessor" variant=" default"/〉
            〈vxbpel:VPChoice vpname=" reply" variant=" default"/〉
            〈vxbpel:VPChoice vpname=" assign" variant=" default"/〉
          〈/vxbpel:VPChoices〉
        〈/vxbpel:RequiredConfiguration〉
      〈/vxbpel:Variant〉
      〈vxbpel:Variant name=" encrypted"〉
        〈vxbpel:VariantInfo〉Information will be decrypted after receipt.
              The customer's privacy is better protected this way.
        〈/vxbpel:VariantInfo〉
        〈vxbpel:RequiredConfiguration〉
          〈vxbpel:VPChoices〉
            〈vxbpel:VPChoice vpname=" receive" variant=" encryption"/〉
            〈vxbpel:VPChoice vpname=" approver" variant=" encryption"/〉
            〈vxbpel:VPChoice vpname=" assessor" variant=" encryption"/〉
            〈vxbpel:VPChoice vpname=" reply" variant=" encryption"/〉
            〈vxbpel:VPChoice vpname=" assign" variant=" encryption"/〉
          〈/vxbpel:VPChoices〉
        〈/vxbpel:RequiredConfiguration〉
      〈/vxbpel:Variant〉
     〈/vxbpel:Variants〉
   〈/vxbpel:ConfigurableVariationPoint〉
 〈/vxbpel:ConfigurableVariationPoints〉
〈/vxbpel:VariabilityConfigurationInformation〉
```

图 12 贷款核准服务中可变性配置的 VxBPEL 规格说明

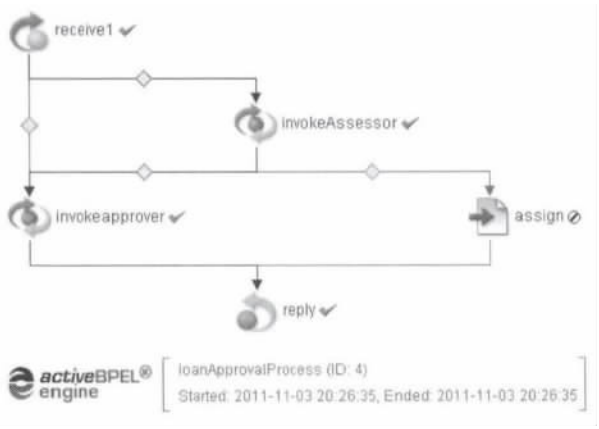

图 13 默认模式工作流场景

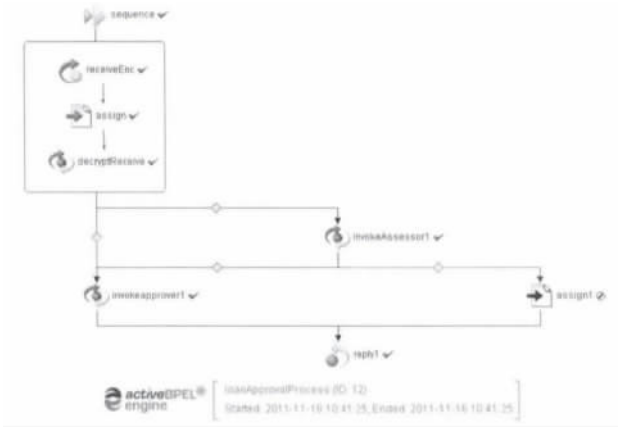

图 14 加密模式工作流场景

4.2　智能货架 SmartShelf

在智能货架系统 SmartShelf [17] 中，客户（Consumer）向 SmartShelf 发送购物请求（包括物品名称和数量），接受请求后 SmartShelf 创建服务时间，记录物品数量，根据物品信息读取物品所属货物类别名称，读取该物品的保质期. 读取结束后，检查物品数量、位置和保质期. 具体说来，（1）检查物品数量时：当货架物品数量满足要求时，停止数量检查；当货架物品数量不足时，通知 WarehouseManager 检查仓库存货：①存货充足时，通知 Carrier 将物品搬运至货架；②存货不足时，通知 Staff 进行进货；③更新产品目录；（2）检测物品位置时：如果物品位置摆放正确，停止检测；否则，调整物品位置；（3）检测物品保质期时：如果没有过期，停止检测；否则，给 Warehouse 发送消息，Warehouse 收到消息后更换货架产品. 当数量、位置、保质期均符合条件时，系统向 Consumer 发送确认消息；否则，向 Consumer 发送相应的失败消息，取消订单.

与贷款核准服务 LoanApproval 相比，智能货架系统 SmartShelf 中的业务流程更加复杂. 一般情形下，订单的处理需要进行多方面的检测；当 Warehouse 存货不足时，无需进行货物位置与过期方面的检测. 采用 BPEL 进行实现上述复杂流程时，服务组装规格说明中将出现多层条件语句的嵌套，存在服务组装规格说明可读性差、维护困难、难于适应变化的业务需求等缺点.

采用 VxBPEL 实现 SmartShelf 可以有效地克服上述不足之处. 具体说来，将条件判断地方设置为变异点，将不同情况下的操作设置为变体，采用可变性配置指明不同变体之间的依赖关系. 采用服务组装实现系统时，创建服务时间、记录物品交易数量、根据物品信息读取物品所属货物类别名称、读取物品保质期等步骤存在不同的处理方式，应设置为变异点. 例如，创建服务时间（变异点）可以有系统自动设置（SetupTime）或人工设置（StaffSetupTime）两种实现方式（即变体）.

图 15 示意了该系统部分变异点的 VxBPEL 设计，其中变异点“CheckQuality”中包含了变异点

```
〈vxbpel:VariationPoint name=" CheckQuantity" 〉
  〈vxbpel:Variants〉
    〈vxbpel:Variant name="quantity" 〉
      〈sequence〉
       〈invoke name=" WarehouseManager" partnerLink=" WarehouseManagerPL"
           portType=" man:WarehouseManagerPT" operation=" WarehouseManager"
           inputVariable=" commodity" outputVariable=" warehouse"〉
       〈/invoke〉
       〈vxbpel:VariationPoint name=" sendorder"〉
         〈vxbpel:Variants〉
           〈vxbpel:Variant name=" default"〉
             〈sequence〉
               〈invoke name=" AlertStaff" partnerLink=" SendOrderToStaffPL"
                portType=" staff:SendOrderToStaffPT" operation=" SendOrderToStaff"
                inputVariable=" commodity" 〉
                〈/invoke〉
               ……
            〈/sequence〉
           〈/vxbpel:Variant〉
           〈vxbpel:Variant name=" sendordertocarrier"〉
             〈sequence〉
               〈invoke name=" AlertCarrier" partnerLink=" SendOrderToCarrierPL"
                 portType=" carrier:SendOrderToCarrierPT"
                 operation=" SendOrderToCarrier" inputVariable=" commodity" 〉
                〈/invoke〉
                ……
             〈/sequence〉
            〈/vxbpel:Variant〉
          〈/vxbpel:Variants〉
        〈/vxbpel:VariationPoint〉
        〈invoke name=" UpdateInventory" partnerLink=" UpdateInventoryPL"
          portType=" update:UpdateInventoryPT" operation=" UpdateInventory"
          inputVariable=" commodity" 〉
        〈/invoke〉
      〈/sequence〉
    〈/vxbpel:Variant〉
    〈vxbpel:Variant name=" default"〉
        ……
    〈/vxbpel:Variant〉
  〈/vxbpel:Variants〉
〈/vxbpel:VariationPoint〉
```

图 15　SmartShelf 系统中基于 VxBPEL 的可变性设计

“sendorder”.作为示例,设置如下 4 种可变性配置方案:

(1)“default”. 货架货物充足、位置摆放正确、未过期.

(2)“staff”. 初始 4 个服务均采用手动完成,货架货物充足、位置摆放正确、未过期.

(3)“insufficient”. 仓库货物不充足,不进行位置检测和保质期检测.

(4)“sufficient”. 货架货物不足、仓库货物充足、位置摆放错误、过期需更换货物.

图 16 示例了可变性配置方案为“insufficient”的 VxBPEL 规格说明. 通过 MX4J 修改可变性配置,依据可变性配置,VxBPELEngine 执行不同的工作流场景.

```
〈vxbpel:VariabilityConfigurationInformation〉
 〈vxbpel:ConfigurableVariationPoints〉
  〈vxbpel:ConfigurableVariationPoint id=" smartshelf" defaultVariant=" default"〉
   〈vxbpel:Name〉smartshelf scheme 〈/vxbpel:Name〉
   〈vxbpel:Rationale〉
        It is possible to configure the smart shelf process to support variability.
    〈/vxbpel:Rationale〉
   〈vxbpel:Variants〉
    〈vxbpel:Variant name=" insufficient"〉
        〈vxbpel:VariantInfo〉Different variation configuration of Smartshelf.
     〈/vxbpel:VariantInfo〉
     〈vxbpel:RequiredConfiguration〉
      〈vxbpel:VPChoices〉
       〈vxbpel:VPChoice vpname=" setuptime" variant=" default"/〉
       〈vxbpel:VPChoice vpname=" setupproductcount" variant=" default"/〉
       〈vxbpel:VPChoice vpname=" readinformation" variant=" default"/〉
       〈vxbpel:VPChoice vpname=" warehouse" variant="quantity" /〉
       〈vxbpel:VPChoice vpname=" sendorder" variant=" default"/〉
       〈vxbpel:VPChoice vpname=" rearrange" variant=" empty" /〉
       〈vxbpel:VPChoice vpname=" returnstatus" variant=" empty" /〉
      〈/vxbpel:VPChoices〉
     〈/vxbpel:RequiredConfiguration〉
    〈/vxbpel:Variant〉
    〈 !--define other variation configuration here.--〉
   〈/vxbpel:Variants〉
  〈/vxbpel:ConfigurableVariationPoint〉
 〈/vxbpel:ConfigurableVariationPoints〉
〈/vxbpel:VariabilityConfigurationInformation〉
```

图 16 “insufficient”可变性配置方案的 VxBPEL 规格说明

4.3 VxBPELEngine 性能评估

为了便于度量 VxBPELEngine 时间性能,对实验进行了简化:(1)为了减少通讯对过程执行时间的影响,服务组装中调用的所有 Web 服务与过程部署在同一台计算机上;(2)为了减少业务过程的复杂性对过程执行时间的影响,简化了 Web 服务的相关操作的实现. 表 1 和表 2 分别总结了实验环境配置与两个实例系统的规模.

表 1 实验环境配置

CPU	内存	硬盘	系统类型	操作系统
2.93×2GHz	2GBytes	1TBytes	32 位	Win7

表 2 实验系统的规模

系统名称	VxBPEL 代码行	服务数量
贷款核准	253	3
智能货架	515	19

为了进一步定量测试与分析可变性部分对引擎性能开销,我们采用标准的 BPEL 与 VxBPEL 分别实现上述两个系统的业务过程,然后比较 ActiveBPEL5.0.2 与 VxBPELEngine 执行上述过程的时间开销. 这里的时间开销是指部署时间与运行时间之和. 运行时间与部署时间都取决于服务组装规格说明的复杂性. BPEL 与 VxBPEL 的规格说明的部署都是一次性完成的,部署时间包括过程定义的读取时间与过程部署时间.

表 3 总结与比较了 ActiveBPEL5.0.2 与 VxBPELEngine 执行贷款核准系统 Loan Approval 的时间开销. 该系统部署了 3 个服务,分别为 LoanApproval、LoanAssessor 和 Encryption. 在不加密(Unencryption)情形下,风险评估的起始值为 1000;在加密(Encryption)情形下,风险评估的起始值调整为 10000. 在不加密情形下,当贷款金额小于 1000 时,仅调用 LoanApproval 服务;当贷款金额大于等于 1000 时,调用 LoanApproval 和 LoanAssessor 服

务.在加密情形下,当贷款金额小于 10 000 时,调用 LoanApproval 和 Encryption 服务;当贷款金额大于等于 10 000 时,调用 LoanApproval、LoanAssessor 和 Encryption 服务. BPEL 过程的部署时间为 20 ms,而 VxBPEL 过程的部署时间为 37 ms.

表 3　采用 BPEL 与 VxBPEL 实现 LoanApproval 的时间开销与性能比较

工作流场景名称	服务数量	BPEL 执行时间/ms	BPEL 时间开销/ms	VxBPEL 执行时间/ms	VxBPEL 时间开销/ms	VxBPEL 时间开销与 BPEL 时间开销之比/%
unencryption_noAssessor	1	557	577	535	572	99.13
unencryption_Assessor	2	569	589	551	588	99.83
encryption_noAssessor	2	570	590	546	583	98.81
encryption_Assessor	3	574	594	562	599	100.84

表 4 总结与比较了 ActiveBPEL5.0.2 与 VxBPELEngine 执行智能货架系统 SmartShelf 的时间开销.该系统部署了 19 个服务,运行时刻按照不同情形调用相应的 Web 服务. BPEL 过程的部署时间为 90 ms,而 VxBPEL 过程的部署时间为 220 ms.

表 4　采用 BPEL 与 VxBPEL 实现 SmartShelf 的时间开销与性能比较

工作流场景名称	服务数量	BPEL 执行时间/ms	BPEL 时间开销/ms	VxBPEL 执行时间/ms	VxBPEL 时间开销/ms	VxBPEL 时间开销与 BPEL 时间开销之比/%
shelfsuff_rightlocation_goodstatus	8	564	654	632	852	130.28
shelfsuff_wronglocation_goodstatus	9	556	646	622	842	130.34
shelfsuff_rightlocation_badstatus	10	562	652	644	864	132.52
shelfsuff_wronglocation_badstatus	11	569	658	665	885	134.50
staff_shelfsuff_rightlocation_goodstatus	8	555	645	635	855	132.56
staff_shelfsuff_wronglocation_goodstatus	9	561	651	622	842	129.34
staff_shelfsuff_rightlocation_badstatus	10	558	648	631	851	131.33
staff_shelfsuff_wronglocation_badstatus	11	575	665	669	889	133.68
warehousesuff_rightlocation_goodstatus	11	571	661	633	853	129.05
warehousesuff_wronglocation_goodstatus	12	577	667	631	851	127.59
warehousesuff_rightlocation_badstatus	13	576	666	648	868	130.33
warehousesuff_wronglocation_badstatus	14	577	667	663	883	132.38
insufficient	9	561	651	640	860	132.10

通过上述两个实例系统,展示了基于可变性设计实现适应性服务组装方法,并且评估了 VxBPELEngine 的性能.实验结果不仅验证了所提适应性服务组装方法的可行性,还表明本文开发的 VxBPELEngine 引擎具备良好的时间性能与可扩展性.具体说来:

(1) VxBPELEngine 在处理不同类型的系统时,均表现出良好的性能.具体说来,在执行贷款核准系统时,可变性引入后的时间开销与标准 BPEL 过程的时间开销相当;在执行智能货架系统时,可变性引入后的时间开销是标准 BPEL 过程的时间开销的 130%左右.进一步分析发现:VxBPELEngine 引擎绝大部分时间用于处理标准的 BPEL 元素(包括读取过程定义、部署过程与执行活动).换言之,VxBPELEngine 引擎尽管增加了可变性定义与可变性配置的读取与处理过程,与标准的 BPEL 元素的处理相比,相关的时间开销非常小.

(2) VxBPELEngine 具有良好的可扩展性.一方面,VxBPELEngine 可以处理简单与复杂的服务组装系统.上述实例系统中,简单的工作流场景仅调用 1 个 Web 服务,复杂的工作流场景调用了多达 14 个 Web 服务.另一方面,通过扩展最新版本的 ActiveBPEL 引擎,VxBPELEngine 能够支持多种 BPEL 版本.

(3) ActiveBPEL 在工业界和学术界得到广泛认可与应用,VxBPELEngine 是通过对 ActiveBPEL 的复用与扩展实现的,继承了 ActiveBPEL 良好的结构与性能,因此 VxBPELEngine 具备较强的实用性.

5　相关工作

下面介绍具有代表性的适应性服务组装方法与引擎,并与本文工作进行比较.

Charfi 和 Mezini 对 BPEL 进行面向 Aspect 的扩充,开发了 AOBPEL[18-19]. AOBPEL 借助面向 Aspect 的编程概念,将业务逻辑作为工作流的主要关注,而将相互交错的其它关注说明为工作流的

Aspect.由于将业务逻辑与表达不同关注的 Aspect 分开说明,在编译和运行时刻,必须将这些分离的组装规格说明编织起来. AOBPEL 改进了 BPEL 的模块化支持,但服务组装规格说明的分散和杂乱导致维护和调试非常困难.

Ezenwoye 和 Sadjadi 提出了一个在 BPEL 过程中增加自治行为的框架 TRAP/BPEL[20]. TRAP/BPEL 将服务组装看成是一个复合服务,并假设 BPEL 用来进行组装的描述.为了增加自治能力,需要对已有的 BPEL 过程进行变换,并增加一个通用代理.通用代理的作用是,当一个服务失败时,采用一个预先定义的或新发现的服务来替换. TRAP/BPEL 通过在运行时刻监控来自通用代理的错误或超时事件,了解对各个服务的调用情况.基于代理机制可以部分解决服务组装的适应性,但存在一些不足之处,如导致了多个版本的服务组装规格说明、必须为各个 Web 服务配备相应的代理.

李刚等人[21]提出了一种能够透明、动态地按需使用的虚拟服务模型及 P2P 引擎.通过虚拟服务组装运算以及服务动态查找算法,虚拟服务模型能够较好地适应服务失效、用户位置移动等网络环境的变化. P2P 引擎对虚拟服务组装提供支持,实现透明地、按需使用服务.虚拟服务模型及 P2P 引擎提升了动态网络环境下应用服务适应性,但受限于对未知环境中服务的查找与组合.

成睿星等人[22]提出了一种基于抽象多重服务范例的适应服务组装方法.该方法用阶层式抽象服务范例的概念构造多重服务范例,提供一种可适应性的相似度测量算法来选择适合调整的服务范例,基于调整运算子的服务范例适应性调整方法将其调整成满足用户需求的服务范例.为了提高适应性调整成功率,服务适应性相似度测量方法在选择服务时考虑了适应性需求,降低了服务调整难度.抽象阶层式服务减少了服务组合代价和人工参与程度,提升了服务组合自动化程度.在服务库服务数量不足、服务组合成功次数有限的情形下,服务组合成功率会大幅度降低.

AdaptiveBPEL[23]是一个将wsBus和面向Aspect技术结合起来的适应性服务组装框架. wsBus[24]是一个消息中间件,引入了一个虚拟终端的概念.由于所有的消息都必须经过虚拟终端进行路由选择,wsBus 可能成为了服务组装运行时刻的瓶颈. AdaptiveBPEL 将不同 QoS 关注表达为 Aspect,运行时刻将 Aspect 代码和 BPEL 过程编织起来,并根据事前定义的策略,提供不同 QoS 的服务组装. AdaptiveBPEL 在为服务组装提供了灵活性的同时,也继承了基于代理机制和基于 Aspect 技术的不足之处.

文献[25]对 BPEL 引擎进行扩展,开发了一个网格服务工作流引擎 BPEL FlowEngine. BPEL FlowEngine 采用分层处理机制,可以调用 Web 服务、网格服务以及网格调度器.引擎执行过程实例时依据 BPEL 标准元素"invoke"属性执行相应的任务. VxBPELEngine 通过名字空间区分标准 BPEL 元素和 VxBPEL 元素,采用不同的解释器对过程元素进行解释.新增的 VxBPEL 解释器负责 VxBPEL 元素的读取与部署,经过 VxBPEL 部署后的过程实例为标准 BPEL 过程.

6 结束语

在复用与扩展标准 BPEL 引擎 ActiveBPEL 基础上,开发了支持 VxBPEL 过程引擎 VxBPELEngine.为了支持服务组装中的可变性设计,VxBPEL 对标准的服务组装语言 BPEL 进行了扩展.由于 VxBPEL 引入新的活动类型,标准的 BPEL 引擎无法执行具有可变性设计的 BPEL 过程,本文讨论了 VxBPELEngine 的设计与实现的关键问题,通过两个实例系统验证了基于可变性设计的适应性服务组装方法的可行性,评估了 VxBPELEngine 的性能.

围绕适应性服务组装方法与支持平台的研究比较活跃.已有方法试图从服务组装实现层探讨服务组装实例的适应性问题,存在不易实现、难于维护、性能差等缺点.采用 VxBPEL 设计与实现的适应性服务组装时,首先通过使用 VxBPEL 对标准的 BPEL 过程中不稳定部分实施可变性设计,并设置各种可变性配置方案. VxBPELEngine 在运行时刻解释与执行可变性配置,实现灵活的业务过程.基于可变性设计的适应性服务组装方法从服务组装规格说明层出发,提供了适应性问题的系统化方案,能够适应开放的业务环境和多变的用户需求,具有易于理解、易于维护、效率高等优点.

进一步的研究工作包括:(1)扩展与增强 VxBPEL,支持服务组装中更多的可变性类型[26]和更复杂的可变性配置方案;(2)集成可变性分析工具、设计工具与 VxBPELEngine 引擎,提供一个完整的基于可变性设计的适应性服务组装支持平台.

致　谢　作者感谢荷兰格罗宁根大学Michiel Koning参与了VxBPELEngine早期版本的开发工作、北京科技大学王可参与了服务组装实例的部分开发工作！

参　考　文　献

[1] Papazoglou M, Traverso P, Dustdar S, Leymann F. Service-oriented computing: A research roadmap. International Journal on Cooperative Information Systems, 2008, 17(2): 223-255

[2] Sun Chang-Ai, Wang Guan, Zhao Yong-Mei. Web service development process framework and case study. Highlights of Sciencepaper Online, 2011, 4(20): 1828-1832(in Chinese)
(孙昌爱，王冠，赵永梅. Web服务的开发过程框架及其实例研究. 中国科技论文在线精品论文，2011，4(20)：1828-1832)

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

[4] OASIS. Web services business process execution language version 2.0. 2007. Available at http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html

[5] Nguyen T, Colman A, Talib M A, Han J. Managing service variability: State of the art and open issues//Proceedings of the 5th Workshop on Variability Modeling of Software-Intensive Systems (VaMoS'11). Namur, Belgium, 2011: 165-173

[6] Charfi A, Mezini M. AO4BPEL: An aspect-oriented extension to BPEL. World Wide Web Journal, 2007, 10(3): 309-344

[7] Casati F, Ilnicki S, Jin L, et al. Adaptive and dynamic service composition in eFlow//Proceedings of the 12th International Conference on Advanced Information Systems Engineering (CAiSE'00). Stockholm, Sweden, 2000: 13-31

[8] Siljee J, Bosloper I, Nijhuis J, Hammer D. DySOA: Making service systems self-adaptive//Proceedings of the 3rd International Conference on Service-Oriented Computing (ICSOC 2005). Amsterdam, Netherlands, 2005: 255-268

[9] Colombo M, Nitto E D, Mauri M. SCENE: A service composition execution environment supporting dynamic changes disciplined through rules//Proceedings of the 4th International Conference on Service-Oriented Computing (ICSOC 2006). Chicago, USA, 2006: 191-202

[10] Koning M, Sun C, Sinnema M, Avgeriou P. VxBPEL: Supporting variability for Web services in BPEL. Information and Software Technology, 2009, 51(2): 258-269

[11] Sun C, Rossing R, Sinnema M, et al. Modeling and managing the variability of Web service-based systems. Journal of Systems and Software, 2010, 83(3): 502-516

[12] Sun C, Aiello M. Towards variable service compositions using VxBPEL//Proceedings of the 10th International Conference on Software Reuse (ICSR 2008). Beijing, China, 2008: 257-261

[13] Sun C, Xue T, Aiello M. ValySeC: A variability analysis tool for service compositions using VxBPEL//Proceedings of the 5th Asia-Pacific Services Computing Conference (APSCC 2010). Hangzhou, China, 2010: 307-314

[14] Jacobson I, Griss M, Jonsson P. Software reuse: Architecture, process and organization for business success//Proceedings of the 8th Israeli Conference on Computer Systems and Software Engineering. Herzliya, Israel, 1997: 86-89

[15] Sinnema M, Deelstra S, Nijhuis J, Bosch J. COVAMOF: A framework for modeling variability in software product families//Proceedings of the Software Product Line Conference (SPLC 2004). Boston, USA, 2004: 197-213

[16] ActiveBPEL Web site. ActiveBPEL Engine Architecture. 2011. Available at http://www.net-gov.com.cn/web/technology/activebpel_struct.htm

[17] Joonseok P, Moon M, Keunhyuk Y. The BCD view model: Business analysis view, service composition view and service design view for service oriented software design and development//Proceedings of the 12th IEEE International Workshop on Future Trends of Distributed Computing Systems (FTDCS 2008). Kunming, China, 2008: 37-43

[18] Charfi A, Mezini M. Aspect-oriented workflow languages//Proceedings of the 14th International Conference on Cooperative Information Systems (CoopIS 2006). Montpellier, France, 2006: 183-200

[19] Charfi A, Mezini M. Aspect-oriented Web service composition with AO4BPEL//Proceedings of the European Conference on Web Services (ECOWS 2004). Erfurt, Germany, 2004: 168-182

[20] Ezenwoye O, Sadjadi S M. TRAP/BPEL: A framework for dynamic adaptation of composite services//Proceedings of the 3rd International Conference on Web Information Systems and Technology (WEBIST). Barcelona, Spain, 2007: 216-221

[21] Li Gang, Ma Xiu-Jun, Han Yan-Bo, Wang Jing. Transparent service composition in dynamic network environments. Chinese Journal of Computers, 2007, 30(4): 579-587(in Chinese)
(李刚，马修军，韩燕波，王菁. 动态网络环境下的透明服务组合. 计算机学报，2007，30(4)：579-587)

[22] Cheng Rui-Xing, Yang Fang-Chun, Su Sen. Service composition based on adaptable revision of multiple service case. Journal of Software, 2008, 19(11): 3011-3022(in Chinese)
(成睿星，杨放春，苏森. 基于多重服务范例适应性调整的服务组合. 软件学报，2008，19(11)：3011-3022)

[23] Erradi A, Maheshwari P. AdaptiveBPEL: A policy-driven middleware for flexible Web services compositions//Proceedings of the International Workshop on Middleware for Web Services (MWS 2005). Enschede, Netherlands, 2005: 5-12

[24] Erradi A, Maheshwari P. wsBus: QoS-aware middleware for reliable Web services interaction//Proceedings of the 2005 IEEE International Conference on e-Technology, e-Commerce and e-Service(EEE 2005). Hong Kong, China, 2005: 634-639

[25] Geng Jia-Bin. A BPEL-based grid workflow engine. Microcomputer Information, 2008, 24(33): 81-82, 111(in Chinese)
(耿佳彬. 一种基于 BPEL 的网格工作流引擎. 微计算机信息, 2008, 24(33): 81-82, 111)

[26] Aiello M, Bulanov P, Groefsema T. Requirements and tools for variability management//Proceedings of the IEEE 34th Annual Computer Software and Applications Conference. Seoul, Korea, 2010: 245-250

SUN Chang-Ai, born in 1974, Ph.D., associate professor. His research interests include software testing, software architecture, and service-oriented computing.

XUE Tie-Heng, born in 1987, M. S. candidate. His research interests include software engineering and service-oriented computing.

HU Chang-Jun, born in 1963, Ph. D., professor, Ph. D. supervisor. His research interests include software engineering and high performance computing.

Background

Increasingly, distributed systems are constructed by orchestrating loosely-coupled Web services which are often deployed in an open and dynamic environment and expose their functionalities through the interfaces. In this context, business processes implemented by such service compositions are expected to be flexible and adaptive enough to cater for rapidly-changing requirements. Unfortunately, the standard version of BPEL is limited to support such an expectation. In our previous work, we have developed the VxBPEL by extending BPEL to support variability within service compositions, and proposed a variability-based adaptive service composition approach using VxBPEL. With the approach, designers can identify the possible changes within service compositions and specify them with alternatives (namely variants of a variation point). In this way, the changes are treated as the first-class objects, and the resulting service compositions are able to be extended and reconfigurable as required. The standard version of BPEL engine is not able to interpret the service composition specifications which are written using VxBPEL because of new elements introduced by VxBPEL.

In this study, we examine key issues to the development of such a VxBPEL engine. We developed the VxBPELEngine by extending the ActiveBPEL, which is a widely-recognized, open-source BPEL engine. Two case studies were conducted to demonstrate how the variability-based adaptive services composition approach can be employed to solve the adaptation issue. The results have clearly demonstrated the feasibility of the proposed approach and the encouraging performance of the VxBPELEngine. Our research opens a new direction of adaptive service compositions. Unlike most of existing work which only considers the adaptation issue at the implementation level and focuses on service composition instances, the variability-based adaptive service compositions approach provides a systematic way to explore the adaptation issue at the specification level. Our approach provides a language for specifying variability within service compositions at design time, and the VxBPELEngine reported here is used to support the interpretation of variability at run-time.

This work is supported by the National Nature Science Foundation of China, the Beijing Natural Science Foundation of China, "Twelfth Five-Year" Key National Science and Technology Program, the Fundamental Research Funds for the Central Universities, and the Beijing Municipal Training Program for Excellent Talents.