

一种规则驱动的网络服务组装机制

孙 熙 刘 譞 哲 焦 文 品 黄 罡 梅 宏

(北京大学信息科学技术学院软件工程研究所 北京 100871)

摘 要 提出了一种规则驱动的服务组装方法,实现了一个基于软件 Agent 的框架,在运行时刻监控和管理组装流程的执行.该方法给出一个算法将流程规约转化为等价的规则集合以用于指导 Agent 的行为,并允许用户通过定义一组可插拔(pluggable)的自适应策略,方便地扩展流程对变化的适应能力.框架实现基于反射式中间件平台 PKUAS,该平台为网络服务和软件 Agent 提供运行支持,并基于其反射机制为 Agent 提供运行时刻的环境信息.

关键词 服务组装;规则;适应性;软件 Agent

中图法分类号 TP311

A Rule-Based Approach to Supporting Adaptable Web Service Composition

SUN Xi LIU Xuan-Zhe JIAO Wen-Pin HUANG Gang MEI Hong

(Institute of Software Engineering, School of Electronic Engineering and Computer Science, Peking University, Beijing 100871)

Abstract Business processes built from Web services need a more adaptable composition solution. In this paper, a rule-driven approach is proposed to control the executions of business processes via agents' behavior rules, which can be generated automatically and modified dynamically to enable the adaptations of business processes. In the approach, the adaptations of the business process are specified in independent adaptation units and agents can load and interpret user-defined adaptation units at runtime. Thus, the executions of business processes can be adapted dynamically. This paper also describes a running support of lightweight agents on a reflective middleware, on which agents can be generated automatically according to the specifications of business processes and the adaptation units to compose Web service to support adaptable business processes.

Keywords Web service composition; rules; adaptation; software agent

1 引 言

网络服务为 Internet 环境下的软件系统的互操作及集成提供了一种新的范型.它平台独立的服务描述以及松耦合的通信模式为不同应用领域的异构系统提供了一种非常灵活的集成手段.现阶段,人们

对基于网络服务的软件系统的集成越来越关注.随着研究的不断深入,网络服务的无缝集成已经在 B2B(商业对商业)的电子商务模式和企业应用集成中显示出巨大潜力^[1].

相比于传统的工作流技术,由一组网络服务组装而成的业务流程需要更加灵活和具有易适应性(adaptability)的组装方案.这是因为,在高度开放、

收稿日期:2006-02-16;修改稿收到日期:2006-04-20.本课题得到国家“九七三”重点基础研究发展规划项目基金(2002CB212003)、国家自然科学基金(60233010,60303004,90412011)资助.孙 熙,男,1982 年生,博士研究生,主要研究方向为软件工程、智能软件、构件技术等.刘譞哲,男,博士研究生,主要研究方向为面向服务体系结构、Web 服务、中间件中的特征交互问题等.焦文品,男,1969 年生,博士,副教授,主要研究方向为软件工程、智能软件、构件技术等. E-mail: jwp@sei.pku.edu.cn.黄 罡,男,1975 年生,博士,副教授,CCF 会员,主要研究方向为分布式计算、中间件技术、软件构件技术、软件体系结构等.梅 宏,男,1963 年生,博士,教授,博士生导师,CCF 高级会员,主要从事软件工程、软件复用和软件构件技术以及分布对象技术等方面的研究.

动态的 Internet 环境下,服务提供者可能随时退出网络或由于网络原因变得不可用;同时,由于商业需求的不断变化,服务流程中涉及的组织可能需要经常变更其商业策略、合作伙伴、合作条件等等.从而,对流程适应动态变化的能力的支持已经成为网络服务组装领域研究的重要话题.

当前描述网络服务组装流程的方法主要可以分为两类,即基于本体论和语义的组装方案以及基于工业标准的组装方法^[2].这两类方案在对动态变化的适应能力上均有所不足.

在工业界,网络服务被视为业务流程的一种标准的抽象接口,服务的集成(或组装)以及服务之间交换消息的流程通过网络服务流程语言来描述.这方面的典型方案为 IBM 与 Microsoft 提出的 BPEL4WS^[3]. BPEL4WS 为每个参与流程的服务指定相应的角色,并基于此给出消息交换的逻辑流程.逻辑上, BPEL4WS 类似于某种编程语言,也提供了诸如顺序(sequence)、分支(switch)、循环(while)等控制结构及相应的控制流.目前,基于 BPEL4WS 的组装方案已经在实际系统中得到了应用,但在得到一定程度认可的同时,它的不足之处也较明显:其通常认为整个组装流程是预定义好的,从而如果运行时刻某个指定的服务变得不可用,往往导致整个流程的中断;并且,对流程中某一部分做出修改后,需要重启整个流程.这些限制对于金融系统、银行系统等应用领域往往是无法接受的.

在学术界,人们不仅仅关注如何来刻画网络服务的行为,对网络服务的语义的研究和探讨已受到了广泛的关注. OWL-S^[4]是一种用于描述网络服务的能力和属性的本体论语言,在此基础上对服务的查找、选择、匹配、验证等组装相关问题的研究已受到了广泛的关注^[5~7].在语义网络服务的基础上,已出现了一些基于人工智能规划系统的网络服务自动组装方案^[8,9].但这些方案具有较明显的不足:一方面,他们的规划往往是在静态封闭的环境下进行的,无法与 Internet 的变化性、开放性和动态性相适应;另一方面,当组装过程涉及到大量的网络服务时,或者在存在大量可供选择的网络服务时,这些方案会出现规模可扩展性不足的问题.

分析现有工作可以看出,当前的解决方案大都关注于如何给出一个无二义的流程规约,而在对过程的执行环境给出约束方面考虑有所欠缺,这使得现在的流程描述语言对于流程在运行时刻环境发生变化时的适应性方面能力较为有限.需要看到,在动

态开放的 Internet 环境下,提供同样服务的提供者可能存在很多,且这些服务提供者在提供相同的服务时,提供服务的方式(即服务的实现方式)也可能是多样化的,所实现的网络服务与外界的交互接口和交互方式都可能会有所不同,如传递的消息参数名不一样,参数的顺序不一样等,从而在集成那些从 Internet 上动态找到的网络服务时,还可能会出现互操作失配的现象^[10,11].因此,当系统用新的服务来替换失效的原有服务时,还需要能够自动消除在服务提供者改变时可能出现的失配问题.

进一步分析 BPEL4WS 可以看到,在其当前规范中,流程的商业决策过程是通过流程控制语句被硬编码于流程逻辑中的,缺少模块化的描述,其主要原因在于它将工作流作为一个整体,这使得业务限制和策略与业务流程变得不可分离.整个业务逻辑被硬编码于单一模块中,导致整个组装规约难于修改、维护和复用,业务逻辑、约束和策略等都很难独立演化,从而难以满足开放、动态环境引发的新的需求.因而,我们需要一组更灵活的机制来对上述的变化提供支持.

因此,要创建 Internet 上基于网络服务的、具有适应性的业务流程,我们认为至少需要解决下列主要问题:

- (1) 当所需的网络服务不存在或不可用时,如何在 not 导致整个流程中断的前提下自动寻找合适的网络服务来替代它完成任务;
- (2) 进一步地,当所选择的替代服务与原有服务之间互操作存在交互失配时,如何消除不一致的交互,实现与原系统的无缝整合;
- (3) 如何清晰地表示流程的商业逻辑、策略并支持运行时刻的加载和修改.

鉴于 BPEL4WS 已成为事实上的工业界标准,并得到了广泛应用,我们也将以 BPEL4WS 为基础,研究如何建立具有可适应性的 BPEL 业务流程.

本文提出的规则驱动的服务组装方法,实现的基于软件 Agent 的框架可用于在运行时刻监控和管理组装流程的执行.软件 Agent 可被定义为可通过感知环境来评估和调整自身行为的自主实体^[12],我们通过将 BPEL4WS 规约的流程转化为驱动 Agent 行为的规则,来提高流程对于变化的适应能力.该方案具体可概括为以下几点:

- (1) 使用软件 Agent 的行为规则来刻画流程的业务策略和控制流程执行.一方面,将商业规则与流程规约相分离,从而更容易理解和维护.另一方面,

流程的变化可以较容易地通过在运行时刻修改相关规则来实现. 为保证 Agent 的行为与流程规约的一致性, 我们实现了一个算法将流程规约映射为等价的 Agent 行为规则集合.

(2) 环境敏感 (context-aware) 的流程适应能力. 我们的方案基于软件 Agent 的概念和反射式中间件获取环境信息, 如网络带宽、系统负载等等. 基于适应性规则, Agent 能够根据环境来调整流程的执行.

(3) 基于适应性策略单元 (adaptation units) 的灵活的扩展方式. 我们的方案允许用户通过指定一组模块化的、可插拔的适应性策略, 来方便地规约流程对变化性的需求, 开发人员也可以通过定义新的适应性策略单元模版, 来较方便地扩展框架的能力.

本文第 2 节我们给出规则驱动的流程组装方法的整体框架描述 (包括将流程规约映射到 Agent 行为规则的转换算法); 在第 3 节中, 我们给出框架的具体实现, 并通过一个例子描述整个方法流程; 第 4 节介绍相关研究并作比较; 第 5 节总结全文并指出下一步的工作.

2 规则驱动的流程自适应框架

在本节中, 首先对本文规则驱动的服务组装框架给出体系结构层的概要说明; 接着, 我们给出从 BPEL4WS 流程规约中提取规则的算法, 并对采用该算法得到的规则集与原流程规约的功能等价性给出证明; 最后, 我们介绍如何通过定义适应性策略单元来扩展这个规则集, 从而增强流程对动态变化的适应能力.

2.1 概述

图 1 给出了本文所述的组装方法的体系结构. 流程定义者需要给出的是流程对应的 BPEL4WS 规范和所需的适应性策略单元. 适应性策略单元封装了流程对动态变化的适应性策略, 例如, 为流程中的某个关键服务提供一个“后备”服务, 在服务失败时自动调用以保障流程的可用性. 在 2.3 节将对适应性策略单元给出更具体的表述.

对组装流程的监控和维护是通过软件 Agent 来完成. 流程定义者提供的输入将通过解释器转换为 Agent 的行为规则集, 用于驱动 Agent 在运行时对流程的监控行为. Agent 通过一组感应器来获取运行时刻来自其它网络服务的消息. 这些消息将被自动转换为 Agent 内建规则引擎所能识别的知识

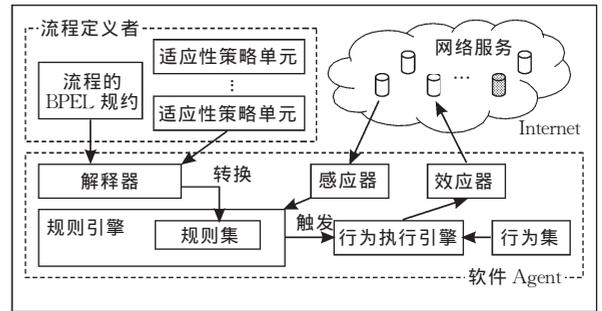


图 1 规则驱动的服务组装框架

对象, 以触发前述的行为规则. Agent 的能力体现为一组预先定义好的行为集合, 包括通过效应器模块来调用某个网络服务, 根据获得的消息更新流程的内部变量, 根据用户策略从一组候选的等价服务中选取一个执行等等. 行为被触发后, 其具体执行过程将由一个行为执行引擎来管理.

2.2 流程规约到规则集的转换

在前文我们已经指出, 流程的 BPEL4WS 规约将作为输入, 被自动转换为规则来实现可适应的工作流. 在本文的框架中, 我们将 Agent 的行为规则分为两类, 一类是从流程规约得到的流程执行规则, 用于控制流程中活动的执行, 本节给出其对应的转换算法; 另一类为适应性规则, 从适应性策略单元给出的适应性策略中转换得到, 用于在需求或环境发生变化时根据给定的策略, 通过选择替代服务等方式调整 workflow, 以应对新的运行状态, 相关讨论将在 2.3 节给出.

我们可以将流程执行规则分为下述两类:

(1) 活动执行规则. 执行规则描述了在组装流程中, 每个活动应该在何时及如何被执行.

(2) 活动依赖规则. 依赖规则体现了流程中活动之间的依赖约束. 在 BPEL4WS 中典型的活动依赖是通过 <link> 标签来描述的同步依赖 (synchronization dependency), 即一个活动只有在所有通过 <link> 标签指定的依赖活动的状态都被确定后, 它才允许被执行 (文献 [3] 对此有更详细的描述).

由于依赖规则的获取可以较容易地通过分析流程规约中的 <link> 标签来得到, 因此在本文中我们主要讨论如何从规约中抽取活动执行规则.

2.2.1 活动执行规则的抽取算法

网络服务流程在流程引擎驱动下的执行过程与程序在计算机上的执行过程是很相似的. 在根据规约描述生成活动规则时, 我们参照程序的执行过程, 规定每个流程 (及后文引入的子流程) 都有一个唯一的标识 (相当于程序代码段的段地址), 每个活动在

流程中都处于某个固定的位置(相当于程序指令的偏移地址)。

在转换算法中,我们首先将活动描述为一个五元组:

活动 := $\langle id, start, finish, next, parent \rangle$.

其中, id 为流程活动的唯一标识,通常为该活动在规约中的名字,如果规约中没有给出,解释器将为其自动生成唯一的名字。

与程序指令的地址描述类似,我们用 $start$ 表示活动在流程中的起始地址, $finish$ 表示活动的结束地址.相应地,复合活动的地址区间将跨越其内部活动的地址区间.流程中第一个活动的 $start$ 值为 1,如果活动的完成意味着流程的完成,则其 $finish$ 值为 -1.

活动的 $next$ 表示该活动执行后流程应转往的地址.对于原子活动,活动的 $next$ 取值与活动的 $finish$ 相同.但当活动为复合活动时,活动内一般包含多个子活动,这时,活动的 $next$ 取值为活动块内的第一个子活动的 $start$ 值.

$Parent$ 给出该活动的上层复合活动的 id .最外层活动的上层活动缺省为流程名.

在上述定义的基础上,我们可以利用如下的算法将流程转换成相应的规则集,其中,过程 2 用来计算各活动对应的 5 元组.

算法 1. 将流程规约转化为行为规则.

过程 1:

1. 由过程 2 得到每个活动对应的元组值.
2. 设置变量 Pid 纪录当前所处的流程标识,其初始值为流程名.设置变量 CA 记录当前处理的活动 id .它们在步 3 生成的规则中将被其实际值所代替.
3. 按序读取流程的规范描述,并按如下所述生成活动对应的规则.其中 $cur_process$ 和 $cur_position$ 为 Agent 内部变量,记录当前流程的 id 和该流程正执行到的地址.

对于原子活动,如 $\langle receive \rangle$, $\langle reply \rangle$, $\langle invoke \rangle$, $\langle assign \rangle$, $\langle terminate \rangle$, $\langle wait \rangle$, $\langle empty \rangle$ 等,产生如下的规则:

$$\frac{cur_process = Pid \ \& \ cur_position = activity.start}{Do(activity); \ cur_position = activity.next}$$

对于复合活动,包括 $\langle sequence \rangle$, $\langle switch \rangle$, $\langle while \rangle$, $\langle flow \rangle$, 根据下列情况产生不同的规则:

- a. 若活动为 $\langle sequence \rangle$ 或 $\langle flow \rangle$, 则

$$\frac{cur_process = Pid \ \& \ cur_position = activity.start}{cur_position = activity.next};$$

- b. 若活动为 $\langle while \rangle$, 且循环条件为 C_w , 则

$$\frac{cur_process = Pid \ \& \ cur_position = activity.start \ \& \ C_w}{Cur_position = activity.next};$$

$$\frac{cur_process = Pid \ \& \ cur_position = activity.start \ \& \ not \ C_w}{Cur_position = activity.finish};$$

- c. 若活动为 $\langle switch \rangle$, 则需要为每一个分支生成一条规则,包括 $\langle otherwise \rangle$ 在内.不妨设活动有 N 个条件选择分支,分支 $i(1 \leq i \leq N)$ 的触发条件为 C_i , 对应活动为 $Activity_i$, $\langle otherwise \rangle$ 对应活动 $Activity_{other}$, 则生成

$$\frac{cur_process = Pid \ \& \ cur_position = activity.start \ \& \ C_i}{Cur_position = Activity_i.start};$$

$$\frac{cur_process = Pid \ \& \ cur_position = activity.start \ \& \ not(C_1 \ and \ C_2 \ and \ \dots \ and \ C_N)}{Cur_position = Activity_{other.start}};$$

4. 如上所得到的规则集即为流程对应的活动执行规则集合.

过程 2:

1. 设置变量 $position$ 记录扫描流程规约过程中的当前位置,其初始值为 1.
2. 按序读取流程的规范描述,如下计算出现的每个活动:

设当前读取到的活动为 α ,

- a. 令 $\alpha.start = position$, $\alpha.next = position + 1$;
- b. 若 α 为 $\langle terminate \rangle$ 活动,令 $\alpha.finish = -1$;
- c. 若 α 为其它原子活动,令 $\alpha.finish = position + 1$;
- d. 若 α 为复合活动,计算该复合活动中包含的活动个数 c ,令 $\alpha.finish = position + c + 1$;
- e. $position = position + 1$.

3. 对步 2 计算出的信息再次遍历并做以下调整:

若 α 是流程的最后一个子活动, $\alpha.next = \alpha.finish = -1$;

若活动 α 是某个复合活动的子活动,

- a. 若活动 α 的父活动为 $\langle flow \rangle$, 令

$$\alpha.start = parent.start + 1,$$

从而,进入 $\langle flow \rangle$ 活动后,所有子活动都将被同时触发;

- b. 若活动 α 的父活动为 $\langle while \rangle$, 令

$$\alpha.finish = parent.start,$$

从而, α 完成后流程将再次回到 $\langle while \rangle$ 活动并评估其条件;

- c. 若活动 α 为父活动的最后一个子活动, $\alpha.next = parent.finish$.

否则,保持步 2 得到的信息不变.

4. 此时所得到的活动元组信息对应于流程的控制结构.

2.2.2 行为规则与业务流程等价性证明

我们说 Agent 的行为规则与业务流程是等价的,如果规则对活动的控制与业务流程对活动的规约是一致的,即:(1)业务流程的执行都能通过规则的执行体现出来;(2)规则的执行不会出现业务流程

不允许的行为。

定理 1. 对服务流程规约运用前述算法,产生的 Agent 行为规则集与该业务流程是等价的。

证明. 考虑到篇幅的限制,我们只证明业务流程中活动的行为方式都能通过特定的规则的执行反映出来。我们可以根据业务流程的结构归纳定义,做如下证明:

(1) 对于任意两个相邻的活动,在规则中,处于前面的活动总在后面的活动之前执行。不失一般性,设这两个相邻的活动都是原子活动 (a_1, a_2)。当 $cur_position$ 的值为 a_1 的起始地址时,规则会执行 a_1 ,并将 $cur_position$ 的值更新为 a_1 的终止地址,实际上就是 a_2 的起始地址。这样根据规则定义,执行 a_2 的规则将被触发。

(2) 对于诸如 `sequence`, `scope`, `switch`, `while` 等复合活动内的活动,规则总是根据相应的复合活动的规范描述控制内部活动的执行。不失一般性,假设复合活动为 $\langle while \rangle$,根据规则,当条件满足时,执行 $\langle while \rangle$ 内的第一个活动,当循环内最后一个活动执行完毕时,规则会将 $cur_position$ 的值变成 $\langle while \rangle$ 块的起始地址,从而使得循环得以继续;否则, $cur_position$ 会指向 $\langle while \rangle$ 块的下一个活动,这样,执行将跳出 $\langle while \rangle$ 块执行之后的活动。故这两条规则与 `while` 的控制语义是等价的。

(3) 对于并发复合活动 $\langle flow \rangle$ 内的活动,根据 `flow` 块的语义, `flow` 内的活动并发执行。在产生规则时,因为所有 `flow` 内的活动的起始地址都相同,且不存在其它条件约束,即它们所对应的规则的前提条件都相同,因此,这些规则将被同时激活,执行对应的活动,即 `flow` 内的活动被并发执行。证毕。

在下一节介绍自适应策略时,引入了子流程 (sub-process) 的概念。对于子流程内的活动,因为每个流程的标识是唯一的,且规则中均给出了对应的流程标识,不同流程的规则不可能同时被触发,即流程对应的规则集是不相交的,根据上面的证明,可以归纳地证明子流程的执行也可以通过规则的执行等价反映出来。

2.3 适应性策略单元

在前文我们已经提到,基于 BPEL4WS 语言描述的流程通常是预定义好且难于演化的。然而,基于网络服务的组装流程处于动态的 Internet 环境中,这使得由固定服务组装而成的服务流程,其可用性很难得到保证。在本文所给出的易适应性组装框架中,流程建模和维护人员可以通过自己设计或使用

预定义好的适应性策略单元,在设计阶段和运行时刻为现有的组装流程添加新的适应性策略,从而支持对流程适应性的设计和维护。适应性策略单元类似于应用程序中插件的概念,用于封装不同的流程适应性策略,并可以在运行时刻加载和删除。目前本框架已实现了若干种适应性策略单元,由于篇幅所限,这里仅介绍其中具代表性的两种,在下文分别称之为“ProcessSwapping”和“ServiceSwapping”。

ProcessSwapping(PS): 对于某个特定的系统目标,流程定义者可以为之定义多个备选的流程,并指定相应的选择策略。例如,可以基于某些服务质量 (QoS) 指标在运行时动态选择某个候选流程。这里我们讨论一类较简单的策略,即按照给定顺序,在当前流程失效时自动选择下一个候选流程,从而提高系统可用性。

ServiceSwapping(SS): 类似地,我们可以定义一组可以互相替代的服务及相应的选取策略。当一个候选服务被选择替代现有服务时,其可能存在的失配现象必须被消除。

为了表述清楚,这里引入一个简单的例子,然后通过分析该实例来展示如何通过适应性策略单元来扩展运行中的组装流程以提供更好的适应性。图 2 给出了一个简单的组装流程“MusicStore”提供音乐的网上购买服务。其首先从购买客户处接收必要的客户信息,如信用卡号、购买歌曲列表等,然后其从歌曲库对应的服务获得 mp3 格式的音乐文件并调用某个转换服务将其转为 wav 格式的文件,最后通过一个刻录服务将这些文件刻录为 CD。

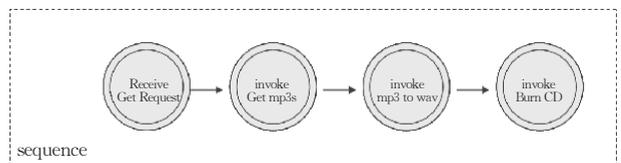


图 2 在线音乐购买服务“MusicStore”的流程

2.3.1 流程的替换

一个特定的目标可以有多个流程以不同的方式加以实现。例如,在上述例子中,如果已知的格式转换服务不很稳定,我们可以定义一种备选方案,通过一个格式转换服务将文件首先转换为 `wmv` 格式,再调用一个 `wmv` 到 `wav` 的转换服务得到符合刻录要求的文件,最后提交给刻录服务。扩展后的流程如图 3 所示。

我们使用 $\langle ProcessSwapping \rangle$ 模版来描述这样一个包含多个候选子流程的活动,其中同样可以使

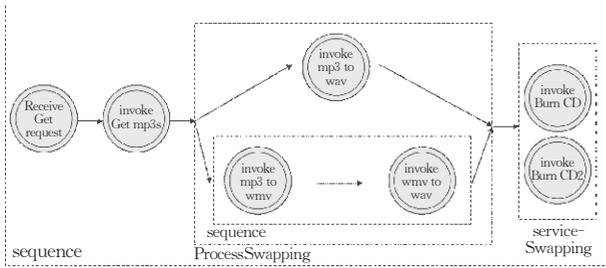


图 3 使用适应性策略单元来扩展流程

用 BPEL4WS 来规约具体的行为. 在该模版基础上, 我们可以较方便地实现所需的适应性策略单元. 在图 4 中, 该适应性策略单元同时封装了最初的“mp3towav”服务和备选流程(“mp3towmv”→“wmvtowav”). 属性 `<target>` 指定了该单元将被应用到原有 MusicStore 流程中的“mp3towav”活动上. `<SelectionPolicy>` 定义了备选流程的选择策略, 具体定义将在后文给出. 当没有一个策略被给出时, 缺省操作将在当前流程失效时自动尝试下一个候选流程. 从而, 当该适应性策略单元被加载后, 流程将在调用“mp3towav”失败后自动执行上述功能等价的子流程.

```

<ProcessSwapping name="PS_MusicConversion">
  <target>"mp3towav"</target>
  <SelectionPolicy>...</SelectionPolicy>
  <candidateProcess name="WF1">
    ...< 调用 mp3 到 wav 转换服务的 BPEL4WS 片段. ->
  </candidateProcess>
  <candidateProcess name="WF2">
    ...< 规约从 mp3 到 wmv,再到 wav 的子流程. ->
  </candidateProcess>
</ProcessSwapping>

```

图 4 ProcessSwapping 模版

2.3.2 服务的替换

通过定义 ProcessSwapping 单元, 流程可以通过不同的子流程来完成同一个目标. 然而在多数情况下, 通常采用的另一种有效方式是将失效的服务替换为另一个功能类似的服务. 对此我们定义 `<ServiceSwapping>` 模版来实现这种策略. 同样对于上述例子, 我们可以通过提供一组 CD 刻录服务来提高整个流程的灵活性和可靠性. 图 5 给出了对应的有多个候选服务的适应性策略单元.

当前的服务替换方案往往假设服务之间可以进行无缝的互操作. 然而在现实情形中, 被调用的替换服务可能由于交互模式(例如消息名、参数顺序等等)与原有服务不同而导致无法工作. 因而要保证系统目标的实现, 即服务流程的顺利执行, 必须保证替

```

<ServiceSwapping name="SS_CDBurning">
  <target>"Burn CD"</target>
  <SelectionPolicy>...</SelectionPolicy>
  <candidateService>invoke name="BurnCDServ2"...</>
</candidateService>
  <candidateService>invoke name="BurnCDServ3"...</>
</candidateService>
</ServiceSwapping>

```

图 5 ServiceSwapping 模版

代服务都能够正确地接受输入参数, 被调用并返回期望的输出结果. 即替代服务能像被替代的服务那样参与 workflow, 对其它那些已存在于 workflow 中的服务来说, 它们可以透明地与替代服务交互, 就像仍然在与被替代的服务交互一样.

但是, 由于服务的替代是在 workflow 执行的过程中动态进行的, 即当发现原有服务无法满足特定的质量要求时或不可用时, 才能确定需要调用特定的替代服务. 因此, 为保证候选服务能被正常调用并成功地提供服务, 必须能够将本来准备传递给原有服务的参数自动转换成候选服务所需的消息. 然而, 不同网络服务的消息的格式及内容的描述的差异可能很大, 很难提供一种全自动化的手段来将针对某个网络服务的消息转换成另一个网络服务所需的并能处理的消息. 另外, 当替代服务需要比被替代服务更多的参数时, 也不可能自动提供或补充替代服务所需的额外信息.

为了实现消息的自动转换, 使得服务的替换过程对用户透明, 我们采取了一种折衷的方案, 该方案基于我们在研究中的两点观察. 首先, 当一个服务具有两到三个较为可靠的替换服务时, 该活动就可以获得令人满意的可靠性; 其次, 用户对于需要的服务(如网上购书)往往使用较为固定的几个提供者, 很少在运行时完全动态地查找并使用某个未知的服务. 因此, 我们的方案允许用户显式给出确定的替代服务, 并在描述信息中指明如何将替代服务的消息映射成替代服务的消息. 在映射时, 可能会出现信息不够或信息多余的情况, 这时, 允许用户为缺少的信息提供某种默认值, 同时将替代服务不需要的信息舍弃.

该消息映射过程在 `<ServiceSwapping>` 单元中, 通过 `<candidateService>` 可选的标签 `<msgMapping>` 来指定. 图 6 给出了相关的描述. 其中, `<inputMessage>` 属性刻画了如何将发送给原始服务的输入消息转换成替代服务的输入消息, 块内的每个 `<part>` 描述了如何将发送给原始服务的消息中的参数名 (source) 的内容赋给替代服务中对应的参数 (desti-

nation). 如果在原始消息中不存在对应的参数,则需要为替代服务的参数指定默认值;如果替代服务不需要该参数,则不需要制定对应的参数. 类似地, `<outputMessage>` 刻画了如何将替代服务返回的结果转换成系统原先所希望得到的结果,即原先规定的网络服务应返回的结果.

```

<ServiceSwapping name="SS_CDBurning">
  <target> "Burn CD" </target>
  <OriginalService>
    <invoke name="BurnCDServ1"/>
  </OriginalService>
  <candidateService>
    <invoke name="BurnCDServ2"/>
    <msgMapping>
      <inputMessage>
        <part source="music",
          destination="songs">
      </inputMessage>
      <outputMessage> ... </outputMessage>
    </msgMapping>
  </candidateService>
</ServiceSwapping>

```

图 6 消除服务的交互失配

2.3.3 选择策略

在上述两类适应性策略单元中,可以通过指定具体的选择策略来在运行时选取最合适的候选服务和子流程,以提供更高的服务质量.

选择策略通过适应性策略单元中的 `<Selection-Policy>` 来指定. 策略表示为一组规则的集合,单条规则使用 `<rule>` 模块来描述. 规则的触发条件可以是 Agent 可感知的环境知识(如网络负载、服务状态等等)或是商业逻辑相关的信息.

例如,在上述流程中,刻录服务配置有多个不同的服务提供者,可以提供不同质量的服务,流程服务决定根据发送请求的用户的身分来决定具体调用的实例;当用户为该流程服务的高级用户(例如, $customerRank > 1$) 时,该流程服务将愿意调用刻录质量较高的服务. `<result>` 给出条件满足时选取的候选服务. 图 7 给出了该规则的对应描述.

```

<selection-policy>
  <rule name="forVIP">
    <condition>
      bpws; getVariableProperty (request,
        customerRank) > 1
    </condition>
    <result>
      <select name="BurnCDServ2">
    </result>
  </rule>
  <rule> ... </rule>
</selection-policy>

```

图 7 选择策略

2.3.4 适应性策略单元到规则的转换

Agent 的行为通过行为规则来进行控制,因而给定的适应性策略单元也需要被转换为相对应的一组规则. 我们将这样的规则称为适应性规则,并实现了从适应性策略单元到适应性规则的转换算法.

为实现对原有流程的调整,我们需要提供能够修改 Agent 现有行为规则的机制,即为 Agent 定义适应性规则. 在适应性规则中,动作部分指定的行为可能是添加、删除某些行为规则等等. 换言之, Agent 的规则系统需要高阶的描述能力.

算法 2. 将适应性策略单元转换为规则.

1. 对于 `<ProcessSwapping>` 单元,假设该 `<ProcessSwapping>` 活动的 id 为 P ,其包含 N 个候选子流程 CP_1, CP_2, \dots, CP_n , C_i 为 CP_i 对应的选择条件.

a. 若用户给出了选择策略,则可直接生成下列适应性策略:

$$\frac{cur_process = Pid \ \& \ cur_position = p.start \ \& \ C_i}{Select \ CP_i \ to \ execute}$$

b. 否则,将选取第一个候选子流程:

$$\frac{cur_process = Pid \ \& \ cur_position = p.start}{Select \ CP_1 \ to \ execute}$$

若当前子流程失效,选取下一个子流程

$$\frac{cur_process = Pid \ \& \ Failed(CP_{i-1})}{Select \ CP_i \ to \ execute}$$

对于 `<ProcessSwapping>` 以外的活动,它的失效意味着它的上层活动失效:

$$\frac{cur_process = Pid \ \& \ Failed(Activity)}{Failed(Activity, parent)}$$

只有当最后一个候选流程失效,该 `<ProcessSwapping>` 活动才失效.

c. 当某个子流程 CP_i 在另一个子流程 CP_j 后被选择时,需要将 $cur_process$ 置为该子流程的 id ,在退出时要恢复 $cur_process$ 原来的值. 为允许子流程的嵌套,我们引入一个堆栈来保存其上层流程的 id 及 $position$ 值:

$$\frac{cur_process = Pid \ \& \ Selected(CP_i) \ \& \ de-Selected(CP_j)}{\begin{array}{l} Delete \ all \ behavior \ rules \ related \ to \ CP_j; \\ Use \ algorithm \ 1 \ to \ generate \ new \ behavior \\ rules \ related \ to \ CP_i; \\ Push \ cur_process \ \& \ cur_position \ into \ stack \\ cur_process = CP_i.id; \ cur_position = 1 \\ \\ cur_process = CP_i.id \ \& \ cur_position = -1 \ \& \\ the \ stack \ is \ empty \end{array}}$$

$$\frac{}{\begin{array}{l} Pop \ the \ parent \ process's \ id \ and \ current \\ position \ out \ of \ the \ stack; \ Assign \ them \ to \\ cur_process \ and \ cur_position, \ respectively \end{array}}$$

2. 对于 `<ServiceSwapping>` 单元,转换方式是类似的,这里不再赘述.

3 组架框架的实现

3.1 对轻量级 Agent 的运行支持

在框架的具体实现中,监控和维护流程运行的 Agent 主要由三部分构成:感应器用于感知环境状态,获取网络服务的调用结果,截取服务间的通信消息,解决失配问题等等;效应器负责根据规则触发的行为去调用服务和更新状态;规则引擎则负责触发和维护行为规则集合.

通过分析可以看出,对于不同的组装流程,感应器和效应器所需要完成的功能基本上是相似的, Agent 只需要加载针对特定流程的行为规则,便可对该流程提供支持;换句话说,用户只需要提供流程的规约,就有可能自动生成其所需要的 Agent. 因此在具体实现时,我们在符合 J2EE 标准的应用服务器 PKUAS 上实现了一套支持轻量级 Agent 的机制,即可以通过加载流程的规约来生成需要的 Agent 实例.

PKUAS 是一个为部署和执行互操作构件提供运行支撑环境的中间件平台^[13]. 在 PKUAS 中,容器负责为构件实例提供运行时环境,同时也可以通过反射机制获取相关构件的所有状态信息. 每个构件实例都对应一组截取器,用来截取和转换外界的服务请求,实现安全、事务等公共服务. 并且,基于 PKUAS 微内核、开放式的结构,可以通过自定义截取器的方式扩展其能力. 感应器获取的信息通过反射式中间件获取,由于本文重点关注于流程适应性及相应 BPEL 模型到规则集的自动转换,这里不再详细阐述,关于 PKUAS 反射体系的具体细节请参见文献^[13].

在本文所述框架的实现中,我们对容器进行了扩展,以实现根据规约生成 Agent 实例以及运行时的支持. 具体架构如图 8 所示.

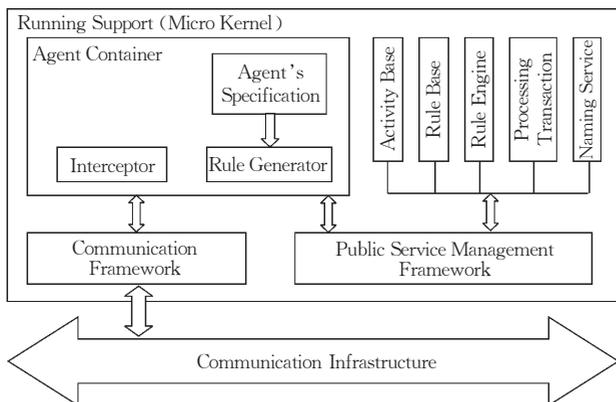


图 8 PKUAS 对 Agent 的运行支撑设施

3.2 对网络服务的运行时支持

Agent 的行为集实现了对网络服务的调用,为了支持组装流程和服务本身的运行,PKUAS 上实现了对网络服务的运行时刻支持^[14],相应的体系结构如图 9 所示.

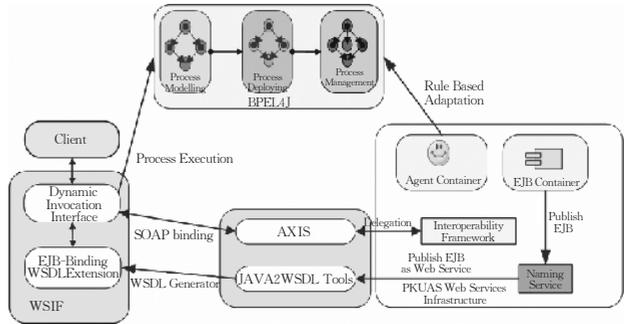


图 9 PKUAS 对于网络服务的支持机制

具体实现上,我们集成了 Apache AXIS^[15] 作为 SOAP 引擎. 通过 AXIS 的 JAVA2WSDL 组件的支持,可以从 Java 类生成对应的 WSDL^[16] 文件. 进一步地,PKUAS 还集成了 WSIF^[17] 框架以支持对 EJB 构件的服务发布和调用,PKUAS 自身的互操作框架用于解析相关的 EJB 绑定信息.

3.3 流程的执行

在本文组架框架的实现中,具有易适应性的流程的执行涉及:(1)流程的启动;(2)流程执行的控制,包括从一个活动切换到另一个活动,通过选择不同的候选子流程来调整流程等等;(3)流程中原子活动的实施;(4)流程的停止.

1. 启动服务流程时, Agent 将内部变量 *cur_process* 和 *cur_position* 分别初始化为该流程名和 1,从而触发了执行第一个活动对应的规则.

2. 当流程被启动后,其运行控制由 Agent 内的规则引擎根据规则触发的行为来驱动执行.

3. 流程的行为最终体现为一组原子活动(即 invoke, receive 等)的执行序列. 相关的调用由 Agent 的效应器完成.

4. 当 *cur_position* 的值为 -1 时,所有规则将不再被触发. 显然,如果流程能顺利执行, Agent 最终必将进入 *cur_position* = -1 的状态,从而结束运行并退出.

3.4 应用实例

仍以网上音乐购买服务 MusicStore 为例,在没有引入适应性策略单元时整个流程将被转换为如下规则集(由于篇幅所限,以下仅列出算法转换得到的规则集合中较重要的规则)

1. *cur_process* = "MusicStore" & *cur_position* = 1 → *cur_position* = 2;

2. *cur_process* = "MusicStore" & *cur_position* = 2 →

```

Do(getMusicList); cur_position=3;
3. cur_process="MusicStore" & cur_position=3 →
Do(getMp3s); cur_position=4;
4. cur_process="MusicStore" & cur_position=4 →
Do(MP3toWAV); cur_position=5;
5. cur_process="MusicStore" & cur_position=5 →
Do(BurnCD); cur_position=-1.

```

当加载前文给出的流程替换单元“PS_Music-Conversion”后, Agent 将生成下列规则来替换原先的规则 4:

```

4.1. cur_process="MusicStore" & cur_position=4 →
deleteRule(4#); Select(WF1);
4.2. Selected(WF1) → Push(cur_process, cur_position);
cur_process="WF1"; cur_position=1;
4.3. cur_process="WF1" & cur_position=1 →
do(Mp3toWav); cur_position=-1;
4.4. cur_process="WF1" & cur_position=-1 →
Pop(cur_process, cur_position); cur_position=5;
4.5. Failed(WF1) → deleteRules(4.1#, 4.2#, 4.3#
and 4.4#); Select(WF2);
4.6. Selected(WF2) → Push(cur_process, cur_position);
cur_process="WF2"; cur_position=1;
4.7. cur_process="WF2" & cur_position=1 →
do(MP3toWMV); cur_position=2;
4.8. cur_process="WF2" & cur_position=2 →
do(WMVtoMAV); cur_position=-1;
4.9. cur_process="WF2" & cur_position=-1 →
Pop(cur_process, cur_position); cur_position=5.

```

当加载服务替换单元“SS_CDBuring”后, Agent 将生成下列规则添加到规则集:

```

5.1. Failed(Burncd1) → Do(Burncd2);
cur_position=-1;
5.2. Failed(Burncd2) → Failed("MusicStore").

```

可见, 随着动态调整行为规则以实现对环境的适应策略, Agent 将能够支持更具适应性的网上音乐购买流程的运行。

4 相关研究工作

Business Rules Group 将规则定义为用于建模商业策略的一组语句。基于规则的方案将业务相关的信息建模为知识库中的知识对象, 将流程的商业策略表示为规则, 用于推导出新的知识和辅助决策。文献[18]提出了一种基于规则的服务流程组装方案, 分抽象定义、规划、构造、执行和演化五个阶段为服务流程的生成提供支持, 然而, 该方案并没有解决查找和组装服务时潜在的交互失配问题, 对于规则之间可能产生的冲突也没有进行讨论。

虽然声明式的规则已被认为是一种自然、灵活

的业务逻辑建模方式, 但在 BPEL4WS 等典型的基于过程的组装语言中, 仍对独立、显式的表示规则缺少支持。当前不少方案如文献[19~21]均关注于通过面向剖面 (Aspect) 技术将规则注入到流程当中。面向剖面技术为模块化系统关注点和运行时刻的操作管理提供了很好的支持机制, 但是这类方案主要的问题在于, 通过其建言 (advice) 的形式引入的规则之间的推理和冲突管理, 必须要由程序员自己来维护, 当规则数量较多时相应的工作是十分困难的。

另一类研究关注于如何在没有预定义流程的情况下, 根据用户的目标动态构造出所需要的流程。文献[6]将用 DAML-S 描述的服务信息转化为 Linear Logic 的公理和规则, 将完成用户目标的组装流程的生成转化为基于该形式化方法的定理证明问题, 并通过一个定理证明器来自动完成。这类工作还包括文献[8,9]等, 它们的局限性在于, 其假设在规划系统进行规划的过程中, 整个外部环境是不会发生变化的, 而 Internet 作为一个具有变化性、开放性、动态性的系统, 随时可能有网络服务退出或加入, 在这方面, 传统的基于形式化证明的规划系统有其不足之处。

研究项目 ADEPT^[22] 和 DESIRE^[23] 等最早给出了基于 Agent 技术对业务流程进行管理的方案, 但其关注的应用主要限于表单处理等变化较少的组织管理流程, 对于动态性较强的应用缺少相应的支持机制。文献[24]给出了一个基于 Agent 的语义网络服务组装框架, 给出了基于能力预期矩阵来评价 Agent 能力与信誉的方案, 同时扩展了合同网协议以满足具有时间限制的任务委托过程; 其重点是关于任务的划分和协商, 而对业务策略变化导致的流程逻辑改变做的工作较少。

文献[25]给出了一个基于 P2P 的动态服务组装模型, 该模型中引入了 Service Container 的概念, 用于封装一系列功能类似的服务, 并根据用户给出的一个效应函数, 在运行时刻选择出最合适的服务实现。本文框架中的适应性策略单元是一个相对更高层的概念, 在具体实现中不仅实现了类似策略的适应性单元, 也可以用于规约其它的流程适应性策略。并且, 与效应函数相比, 基于规则给出的选择策略更加灵活, 同时也更便于修改和维护。

通过分析现有的研究工作, 我们认为多数解决方案仅对流程变化性的一到两个方面提供支持, 缺少对本文前言中所分析的流程变化性的各个方面的整体考虑。因此我们的研究试图给出一种更完整, 易扩展和更具可用性的解决框架, 即以声明式规则来

支持流程的适应性,以类插件的方式扩展现有的工业规范 BPEL4WS,以支持运行时刻动态修改流程逻辑,对失效的服务进行替换,消除潜在的交互失配等。

5 结束语

本文对网络服务组装流程关于变化的适应性问题进行了初步探讨,提出了一种规则驱动的服务组装框架。在该框架的实现中,业务流程的适应性被规约为独立的适应性策略单元,与流程本身均被转换为等价的 Agent 行为规则,从而驱动 Agent 在运行时刻监控和根据需求和环境的变化调整流程逻辑。由于流程业务逻辑的变化可能发生在流程和服务两个层面,我们通过开发两类自适应模板以支持在运行时刻使用候选服务替换流程中的单个服务或整个子流程,以及对选择策略的在线修改。基于框架本身易扩展的架构,可以较方便地开发新的模板以实现更多的适应性策略。

由于我们的框架支持对于业务逻辑的动态修改,从而,需要提供对这种修改的运行时刻验证,以检查修改是否会对流程产生不利影响,如死锁等等。在我们当前的框架中,适应性策略单元本身提供了一个较好的粒度以将流程变化限制于一个可控的范围。以〈ServiceSwapping〉为例,可以保证只要其内部状态是安全的且各候选服务是功能等价的,则修改后的流程仍将保持原有的属性。目前已有部分研究讨论网络服务流程的验证问题^[26,27],然而就我们了解的工作范围中,仍没有在运行时刻检测流程性质的成熟方案。在将来的工作中我们将考虑借鉴动态工作流等领域的研究工作,采用形式化方法对运行时刻修改的验证作进一步的探索。

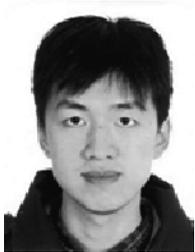
未来工作还包括:(1)对框架实现的性能做进一步的分析和评估;(2)定义更多的适应性策略;(3)提供图形化的适应性策略单元定制界面等。

参 考 文 献

- 1 Milanovic N., Malek M.. Current solutions for Web service composition. *IEEE Internet Computing*, 2004, 8(6): 51~59
- 2 Talib M. A., Yang Z. K., Ilyas Q. M.. Modeling the flow in dynamic Web services composition. *Information Technology Journal*, 2004, 3(2): 184~187
- 3 Curbera F. *et al.* Business process execution language for Web services. Version 1.1, <http://www.ibm.com/developworks/library/ws-bpel/>, 2003
- 4 OWL Services Coalition. OWL-S: Semantic markup for Web services, <http://www.daml.org/services/owl-s/1.0/>, 2003

- 5 Maximilien E. M., Singh M. P.. A framework and ontology for dynamic Web services selection. *IEEE Internet Computing*, 2004, 8(5): 84~93
- 6 Rao J., Kungas P., Matskin M.. Logic-based Web services composition: From service description to process model. In: *Proceedings of the IEEE International Conference on Web Services (ICWS)*, San Diego, California, 2004, 446~453
- 7 Sycara K., Paolucci M., Soudry J., Srinivasan N.. Dynamic discovery and coordination of agent-based semantic Web services. *IEEE Internet Computing*, 2004, 8(3): 66~73
- 8 McIlraith S., Son T.. Adapting Golog for composition of semantic Web services. In: *Proceedings of the 8th International Conference on Knowledge Representation and Reasoning*, Toulouse, France, 2002, 482~496
- 9 Wu D., Parsia B., Sirin E., Hendler J. A., Nauet D. S.. Automating DAML-S Web services composition using SHOP2. In: *Proceedings of the 2nd International Semantic Web Conference*, Sanibel Island, FL, USA, 2003, 195~210
- 10 Jiao W., Mei H.. Eliminating mismatching connections between components by adopting an agent-based approach. In: *Proceedings of the 15th IEEE International Conference on Tools with Artificial Intelligence*, Sacramento, California, 2003, 358~365
- 11 Yellin D. M., Strom R. E.. Protocol specifications and components adaptors. *ACM Transactions on Programming Languages and Systems*, 1997, 19(2): 292~333
- 12 Wooldridge M., Jennings N. R.. Intelligent agents: Theory and practice. *The Knowledge Engineering Review*, 1995, 10(2): 115~152
- 13 Huang G., Mei H., Yang F.. Runtime software architecture based on reflective middleware. *Science in China (Series E)*, 2004, 47(2): 121~138
- 14 Huang G., Liu X., Mei H.. SOAR: Towards dependable service-oriented architecture via reflective middleware. *International Journal of Simulation and Process Modeling*, to appear in January, 2007
- 15 AXIS Version 1.1 rc2. Apache Web Services Project. <http://ws.apache.org/axis>
- 16 W3C: Web Services Description Language (WSDL), Version 2.0. Part 1: Core Language, W3C Working Draft, <http://www.w3.org/TR/wsdl20>, 2003
- 17 WSIF Version 1.0 rc1. Apache Web Services Project. <http://ws.apache.org/wsif/>
- 18 Orriens B., Yang J., Papazoglou M. P.. A framework for business rule driven service composition. In: *Proceedings of the Technologies for E-Services, 4th International Workshop, TES 2003*, Berlin, Germany, 2003, 14~27
- 19 Charfi A., Mezini M.. Hybrid Web service composition: business processes meet business rules. In: *Proceedings of the 2nd International Conference on Service Oriented Computing (IC-SOC)*, New York, USA, 2004, 30~38
- 20 Courbis C., Finkelstein A.. Towards aspect weaving applications. In: *Proceedings of the 27th International Conference on Software Engineering (ICSE 2005)*, St. Louis, Missouri, USA, 2005, 69~77
- 21 Verheecke B., Cibran M., Jonckers V.. Aspect-oriented pro-

- gramming for dynamic Web service monitoring and selection. In: Zhang Liang-Jie ed. *Web Services, European Conference (ECOWS)*. Lecture Notes in Computer Science 3250. Berlin, Germany: Springer-Verlag, 2004, 15~29
- 22 Alty J. A. , Griffiths D. , Jennings N. R. , Mamdani E. H. , Struthers A. , Wiegand M. E. . ADEPT: Advanced decision environment for process tasks: Overview & architecture. In: *Proceedings of the BCS Expert Systems Conference, Applications Track, ISIP Theme*, 1994, 359~371
- 23 Brazier F. M. T. , Jonker C. M. , Treur J. , Wijngaards N. J. E. . Compositional design of a generic design agent. *Design Studies Journal*, 2001, 22(5): 439~471
- 24 Ermolayev V. , Keberle N. , Plaksin S. . Towards a framework for agent-enabled semantic Web service composition. *International Journal of Web Services Research*, 2004, 1(3): 63~87
- 25 Benatallah B. , Sheng Q. Z. , Dumas M. . The Self-Serv environment for Web services composition. *IEEE Internet Computing*, 2003, 7(1): 40~48
- 26 Camara J. , Canal C. , Cubo J. . Issues in the formalization of Web service orchestrations. In: *Proceedings of the 2nd International Workshop on Coordination and Application Techniques for Software Entities (WCAT'05)*, Glasgow, Scotland, 2005, 17~24
- 27 Fu X. , Bultan T. , Su J. . Formal verification of e-services and workflows. In: *Proceedings of the Workshop on Web Services, e-Business, and the Semantic Web (WES): Foundations, Models, Architecture, Engineering and Applications*, Toronto, Canada, 2002, 188~202



SUN Xi, born in 1982, Ph. D. candidate. His current research interests include software engineering, software component technology and intelligent software systems.

LIU Xuan-Zhe, Ph. D. candidate. His research interests are in service oriented architecture (SOA), Web services, feature interactions in middleware-based systems.

JIAO Wen-Pin, born in 1969, Ph. D. , associate profes-

or. His current research interests include software engineering, software component technology, and intelligent software systems.

HUANG Gang, born in 1975, Ph. D. , associate professor. His research interests are in distributed computing, middleware, component technology, software architecture.

MEI Hong, born in 1963, Ph. D. , professor, Ph. D. supervisor. His current research interests include software engineering and software engineering environment, software reuse and software component technology, distributed object technology.

Background

This work is supported by the National Basic Research Program of China (973 Program) under grant No. 2002CB212003; the National Natural Science Foundation of China under grant Nos. 60233010, 60303004, and 90412011. The research scopes cover several fields of software engineering, including software reuse and software component technology, domain engineering, software architecture, component operating platform, software agent technology, Internetware, etc.

The team has made important progress, amongst of which is the proposition of Architecture-Based Component-Oriented (ABC) software development approach. The ABC approach proposes to offer an effective systematic solution for

component-based reuse by taking advantage of both software architecture and component-based software development. Several tools and platforms have been developed, e. g. the component operating platform PKUAS. Some tools are already applied in industrial projects.

The work described in the paper is aimed at composing Web services to support adaptable business processes. In the framework and implementation, the adaptations of the business process are specified in independent adaptation units, and then process specification and the units will be interpreted into behavior rules and loaded into an agent at runtime. Thus, the executions of business processes can be adapted by the agent dynamically.