一种策略驱动的BPEL流程异常处理框架

王权于[1] 吕国斌[1] 应 时[2] 周 峰[1]
(中国地质大学(武汉)网络教育学院 武汉 430074)[1] (武汉大学软件工程国家重点实验室 武汉 430072)[2]

摘 要 如何提高BPEL流程异常处理的开发效率是策略驱动的BPEL流程异常处理方法亟待解决的关键问题之一。首先分析了基于策略的BPEL流程异常处理机制,设计了一种新的BPEL流程异常处理策略描述语言BPEH/PDL,然后结合BPEH/PDL异常处理策略,给出了一种新的BPEL流程异常处理框架BPEH/F,它具有一定的应用意义。

关键词 BPEL流程,异常处理,策略,框架
中图法分类号 TP311 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2015.1.041

Policy Driven Exception Handling Framework for BPEL Processes

WANG Quan-yu[1] LV Guo-bin[1] YING Shi[2] ZHOU Feng[1]
(Distance Education College, China University of Geoscience, Wuhan 430074, China)[1]
(State Key Lab of Software Engineering, Wuhan University, Wuhan 430072, China)[2]

Abstract How to improve the efficiency of the development of BPEL processes exception handling is one of the key issues for policy-driven exception handling of BPEL processes to be sovled. Firstly, the paper analysized the exception handling mechanism of BPEL processes, creatively designed BPEH/PDL language, a new Policy Description language (PDL) for exception handling of BPEL processes, and then based on BPEH/PDL language, proposed the exception handling framework BPEH/F, a integrated BPEL processes exception handling framework.

Keywords BPEL processes, Exception handling, PDL, Framework

1 引言

异常处理作为一种重要的软件容错机制,在程序设计语言和工作流等方面已得到广泛而深入的研究和应用。Web服务的分布自治性、BPEL流程结构的松散耦合性和运行环境的动态不确定性,使得如何提高BPEL流程的容错能力成为面向服务软件工程的科学问题之一。尽管WS-BPEL语言提供了内置的异常处理机制[1]以支持BPEL流程的异常处理,但BPEL规范支持BPEL流程的容错能力还存在着诸多值得研究的问题:如何实现BPEL流程异常处理逻辑和正常业务逻辑关注点分离,从而提高BPEL流程异常处理逻辑的可复用性和灵活性;如何形式化规约和验证BPEL流程的异常处理逻辑正确性,正确地指导和控制BPEL流程异常处理逻辑的行为;如何高效地实现BPEL流程的异常处理逻辑,提高BPEL流程异常处理的开发效率等等。

基于策略的管理技术已广泛应用于网络、信息安全和分布式系统等领域[2]。策略驱动的BPEL流程异常处理方法通过策略规范BPEL流程的异常处理逻辑,有效地分离了BPEL流程的正常业务逻辑和异常处理逻辑;基于BPEL流程异常处理策略语言的形式化语义,建立BPEL流程异常处理策略的形式化模型,分析和验证BPEL流程异常处理策略的潜在冲突,能正确地控制BPEL流程异常处理行为;设计基于策略的BPEL流程异常处理框架,支持BPEL流程异常处理策略的实现,提高BPEL流程异常处理逻辑的开发效率,进而提高BPEL流程的容错能力,改善BPEL流程可信性。

本文充分地分析了已有BPEL流程异常处理研究工作,针对如何提高BPEL流程异常处理开发效率这个亟待解决的关键问题,提出了一种新的策略驱动的BPEL流程异常处理框架。本文第2节介绍相关工作;第3节介绍BPEL流程异常处理机制;第4节介绍一种新的BPEL流程异常处理策略描述语言——BPEH/PDL;第5节介绍BPEH/PDL策略驱动的BPEL流程异常处理框架BPEH/F;最后总结全文。

2 相关工作

Zeng等人将基于策略的BPEL流程异常处理框架分为集中型(Integration Approach)和分离型(Separation Approach)[3]。集中型的基本思想:BPEL流程设计时,正常业务逻辑和异常处理逻辑开发人员分别设计正常业务逻辑和基于策略的异常处理逻辑。流程部署阶段,通过部署工具(如BPEL流程集成器)将表示正常业务逻辑的WS-BPEL流程和表示异常处理逻辑的异常处理策略集成,生成具有容错能力的标准的WS-BPEL流程,并部署到标准的BPEL流程引擎

到稿日期:2014-03-07 返修日期:2014-05-13 本文受国家自然科学基金项目(61070012),湖北省自然科学基金项目(2013245080)资助。
王权于(1971—),男,博士,副教授,CCF高级会员,主要研究方向为SOA、语义Web服务;吕国斌 教授,主要研究方向为网络安全;应 时教授,博士生导师,主要研究方向为SoA、云计算;周 峰 硕士生。

解释执行。由于在 BPEL 流程部署阶段生成了容错 BPEL 流程，因此框架支持部署时 BPEL 流程异常处理策略的实现。分离型 BPEL 流程异常处理方法的基本思想：将表示正常业务逻辑的 BPEL 流程和表示异常处理逻辑的异常处理策略分开部署，即标准 BPEL 引擎负责 BPEL 流程正常业务逻辑执行，策略引擎负责 BPEL 流程异常处理策略的执行，通过事件机制或发布/订阅机制实现 BPEL 流程引擎和策略引擎间交互。BPEL 流程运行时若发生异常事件，则激活相应的异常处理策略，指导和控制流程引擎或异常处理设施完成异常处理。分离型框架是在 BPEL 流程运行发生异常时，即时地通过 BPEL 流程异常处理策略来指导和控制 BPEL 流程的异常行为，所以将其称为运行时 BPEL 流程异常处理策略框架。

A. Charfi 等人设计了一种集中型的 BPEL 流程异常处理框架原型体系结构[4]。BPEL 流程设计阶段，流程模型库和异常处理策略库分别对基于 WS-BPEL 的正常业务逻辑和基于策略的 BPEL 流程异常处理逻辑持久化，实现了正常业务逻辑和异常处理逻辑的分离。BPEL 流程部署阶段，流程模型重构模块执行模型重构算法，将异常处理策略转换为标准的 WS-BPEL 代码片段插入到相应的 BPEL 流程，BPEL 流程执行规划模块使用本地优化或全局规划，为生成的容错 BPEL 流程选择最佳的伙伴服务。生成的容错 BPEL 流程部署到 BPEL 引擎并被执行。刘安等人提出了一种容错的事务性 BPEL 流程框架 FACTS[5]，FACTS 为容错的服务组合提供了一个集成的规约异常处理规约模块（Specification Module）、验证（Verification Module）和执行环境（Implementation Module）。Specification Module 基于 ECA 规则范型，提供了异常处理逻辑的设计支持；Verification Module 验证 BPEL 流程异常处理逻辑的正确性；Implementation Module 负责将表示异常处理逻辑的 ECA 规则转化为标准的 WS-BPEL 代码，将转化后的 WS-BPEL 代码插入到正常业务逻辑中。A. Erradi 等人基于 WS-Policy4MASC 策略描述语言，提出了一种策略驱动的可管理和自适应的服务组合中间件 MASC[6,7]。MASC 利用 SOAP 消息层和业务流程编制层的协同管理，实现了跨层的 BPEL 流程异常处理。MASC 框架自上而下分为 Web 服务组合层、服务组合适应层、绑定和 Web 服务调用层、SOAP 消息层和资源层。Web 服务组合层基于 Microsoft Workflow Foundation（WF），实现 BPEL 流程正常业务逻辑的编制；服务组合适应层由事件监控器（Event Monitor）、策略决策点（PDP）和作为策略执行点（PEP）的适应管理器（Adaptation Manager）组成。MASC 将等价服务组织成虚拟端点（VEP），Web 服务选择管理器按照等价服务选择策略，基于收集到的 Web 服务 QoS 信息，在已注册的虚拟端点（VEP）中选择 BPEL 流程的伙伴服务。A. Charfi 等人基于 AO4BPEL 语言，利用方面（Aspect）来表示服务组合的自适应特征，提出了一种自适应的服务组合体系结构[8]。该方法支持适应能力的扩展和插件的热部署，需要改造标准的 BPEL 流程引擎。D. Karastoyanova 等人基于 BPEL 流程事件模型[9]、发布/订阅模式、AOP 技术和 WS-Policy 框架，提出了一种面向方面的 BPEL 流程异常处理框架[10]。BPEL 流程异常处理建模为横切关注点，采用异常处理服务来执行异常处理动作，定义了 BPEL 流程异常处理方面断言语言，采用 WS-Policy 框架描述基于方面的 BPEL 流程异常处理策略。

3 基于策略的 BPEL 流程异常处理机制

BPEL 流程由多个原子活动或结构化活动按照 WS-BPEL 规范组合而成，原子活动或结构化活动调用伙伴服务实现业务逻辑。

3.1 异常类型

BPEL 流程异常是指 BPEL 流程运行时，由于 BPEL 流程、伙伴服务和运行环境等故障导致 BPEL 流程行为偏离了正常执行路径。从异常的故障来源角度，BPEL 流程异常分为系统异常、资源异常和应用异常（见图 1）。

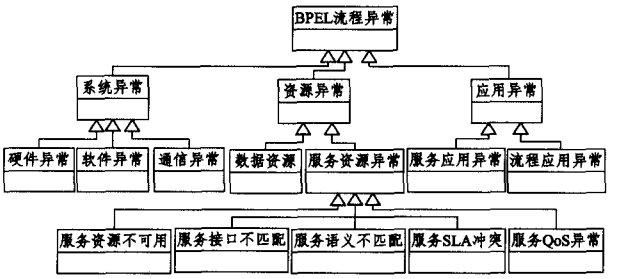

图 1 BPEL 流程异常分类

系统异常（System Exception）是指 BPEL 流程运行时因软硬基础设施故障引发的异常。系统异常分硬件异常、软件异常和通信异常。硬件异常是指服务器、存储等设备故障引发的异常，这类异常可通过硬件冗余（如集群技术）来预防。一旦出现硬件异常，必须挂起或终止 BPEL 流程实例的运行，由人工参与处理。软件异常是指由于操作系统、数据库、中间件系统等软件故障引发的异常，软件异常可通过软件冗余预防，有时也需要人工干预处理。通信异常是指由于网络故障导致资源访问失败，通信异常可通过链路冗余预防，对于暂时性通信异常，可通过重试模式处理。

资源异常（Resource Exception）是指 BPEL 流程运行时，由于数据资源、计算资源等故障引发的异常。BPEL 流程计算资源是指其伙伴服务，引发资源异常主要是伙伴服务故障导致，BPEL 流程资源异常通常指服务资源异常。资源异常分为服务资源不可用、服务资源内容异常、服务接口不匹配、服务协议绑定错误、服务 QoS 异常、SLA 冲突。

应用异常（Business Exception）是指伙伴服务或 BPEL 流程应用逻辑设计缺陷引发的异常。应用异常分为服务应用异常和流程应用异常。服务应用异常是指不满足服务业务协议引发的异常。流程应用异常是指流程运行时上下文信息不满足流程应用逻辑或约束引发的异常。应用异常是用户自定义异常，由 Web 服务和 BPEL 流程设计人员设计阶段定义。但设计阶段预测所有的应用异常是不现实的，参照文献[11]中的方法，定义 AnonymousException 表示设计阶段不可预测异常。

3.2 异常处理动作

异常处理动作（EHAction）是对 BPEL 流程异常的反应。异常处理动作分原子动作（Atomic Action）和复合动作（Composite Action）。原子动作不可再分，表示基本的异常处理行为；复合动作由原子动作或复合动作按不同控制结构组合而成，以支持复杂的异常处理逻辑表示。

3.2.1 原子动作

原子动作是不可再分的 BPEL 流程异常行为，是对常见 BPEL 流程异常处理逻辑的抽象。引发异常的 scope 故障作用域生命周期如图 2 所示。scope 开始是就绪状态(Inactive)，等待 BPEL 流程引擎激活；scope 被激活后进入运行状态(Running)；scope 运行过程中可能进入多种状态：①完成状态(Completed)：scope 顺利完成流程逻辑，BPEL 流程引擎继续后续活动执行；②故障状态(Faulting)：scope 活动由于运行环境、流程设计或伙伴服务资源等故障抛出异常，scope 活动开始异常处理；③终止状态：scope 终止 BPEL 流程实例的运行；④补偿状态：嵌套 scope 的父 scope 可补偿子 scope 活动，保证 BPEL 流程的状态一致性。

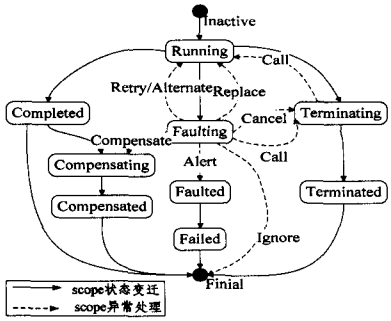

图 2　scope 活动状态

结合 BPEL 流程异常处理需要，本文原子动作分为忽略(Ignore)、跳过(Skip)、重试(Retry)、替代(Alternate)、替换(Replace)、流程取消(Cancel)、补偿(Compensate)、调用(Call)、报警(Alert)等 9 类(见表 1)。为更好描述原子动作，先给出几个相关基本定义：

定义 1(Web 服务)　设 OP 为 Web 服务操作集合：$OP=\{op_1,op_2,\cdots,op_n\}$，$QoS$ 为 Web 服务质量属性集：$QoS=\{QoS_1,QoS_2,\cdots,QoS_n\}$，Web 服务是一四元组 $WS=(Name,Desc,OP,QoS)$，其中 $Name$ 为 Web 服务名，$Desc$ 为服务描述，OP 表示服务功能属性，QoS 表示服务非功能质量属性。

定义 2(相容服务)　设 WS_1 和 WS_2 属于 Web 服务，如果 WS_1 和 WS_2 功能相容，即 WS_1 操作集合包含 WS_2 或 WS_2 的操作集合包含于 WS_1，则 WS_1 和 WS_2 为相容服务，记为 $WS_1\sim WS_2$。

定义 3(等价服务)　设 WS_1 和 WS_2 属于 Web 服务，如 WS_1 和 WS_2 是相容服务，且 WS_1 和 WS_2 具有相同非功能属性，即 WS_1 的 QoS 等于 WS_2 的 QoS，则 WS_1 和 WS_2 是等价服务，记为 $WS_1\approx WS_2$。

定义 4(服务社区)　设 $WSC_1,WSC_2,\cdots,WSC_n$ 为相容服务，服务社区 $WSC=\{WSC_1,WSC_2,\cdots,WSC_n\}$ 是相容服务集，记为 $WSC(WSC_1,WSC_2,\cdots,WSC_n)$。

根据原子动作对故障作用域状态的改变，将原子动作分为①强制类动作(DA)：DA 动作直接将故障域状态从故障变为结束状态(完成、取消和补偿状态)，$DA=\{$Ignore, Skip, Cancel, Compensate$\}$；②尝试类动作(RA)：RA 类动作采用冗余机制处理异常，$RA=\{$Retry, Alternate, Replace$\}$；③其他动作：这类动作包括外部调用动作和报警动作，$NA=\{$Call, Alert$\}$。原子动作关系如图 3 所示。

表 1　BPEL 流程异常处理动作

原子动作	执行语义
Ignore	BPEL 流程忽略异常，继续执行 scope 后继活动。scope 内所有活动被强制结束，BPEL 流程由故障状态直接回到完成状态。
Skip	BPEL 流程跳过 scope 后继活动 As 继续执行。从控制流的角度来看，BPEL 流程跳过 As 而转移到 As 的后继活动 Ap 继续执行。
Retry	scope 抛出异常时，BPEL 流程重复执行 scope，直到活动成功完成，或者达到设定的重试次数或截止时间。
Alternate	从服务社区$\{WS_1,WS_2,\cdots,WS\}$依次绑定等价服务，直到成功或所有等价服务执行完毕。
Replace	如活动 $Activity_1$ 和 $Activity_2$ 为等价活动，当故障域 $Activity_1$ 发生异常时，可用 $Activity_2$ 代替 $Activity_1$ 执行。
Compensate	故障域发生异常时，如果其内嵌的 $scope_1,scope_2,\cdots,scope_n$ 已经完成，则按照逆序分别补偿 $scope_n,scope_{n-1},\cdots,scope_1$。
Cancel	立即取消故障活动的执行。
Call	调用外部服务(如挂起和恢复等)处理 BPEL 流程实例异常。
Alert	通知 BPEL 流程管理人员发生的异常或将异常消息记录到日志。

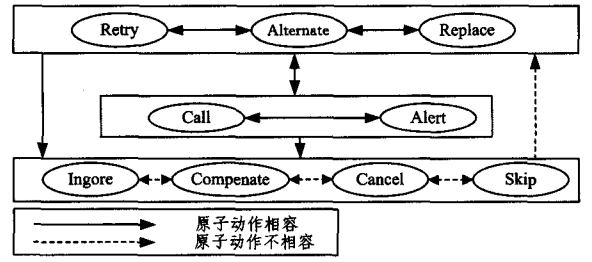

图 3　异常处理原子动作关系

由原子动作关系得到其相容性质如下：

性质 1　如果 $a1,a2$ 都是尝试类动作，即 $a1,a2\in RA$，那么 $a1$ 和 $a2$ 相容，记为：$a1\sim a2$。其形式化表示为：$a1,a2\in RA\rightarrow(a1\sim a2)\wedge(a2\sim a1)$。

性质 2　如果 $a1,a2$ 都是其他类动作，即 $a1,a2\in NA$，那么 $a1$ 和 $a2$ 相互相容，记为：$a1\sim a2$。即：$a1,a2\in NA\rightarrow(a1\sim a2)\wedge(a2\sim a1)$。

性质 3　如果 $a1$ 是尝试类动作，$a2$ 为其他类动作，那么 $a1$ 和 $a2$ 相容，$a2$ 和 $a1$ 相容。即：$a1\in RA,a2\in NA\rightarrow(a1\sim a2)\wedge(a2\sim a1)$。

性质 4　如果 $a1$ 是尝试类动作，$a2$ 为强制类动作，那么 $a1$ 和 $a2$ 相容，但 $a2$ 和 $a1$ 不相容。其形式表示为：$a1\in RA$，$a2\in DA\rightarrow(a1\sim a2)\wedge\neg(a2\sim a1)$

性质 5　如果 $a1$、$a2$ 都是强制类动作，那么 $a1$ 与 $a2$ 不相容，即 $a1\in DA,a2\in DA\rightarrow\neg(a1\sim a2)\wedge\neg(a2\sim a1)$。

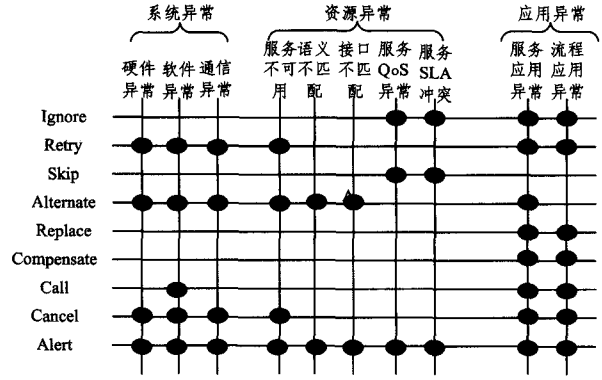

图 4　BPEL 流程异常-动作关系矩阵

通过分析 BPEL 流程异常模型和异常处理动作得到 BPEL 流程异常-动作矩阵关系(见图 4)以指导异常设计人员

开发 BPEL 流程异常处理策略。

3.2.2 复合动作

复合动作是由原子动作或复合动作按不同控制结构组合而成，能处理复杂的 BPEL 流程异常处理逻辑。目前 BPEH/PDL 语言定义的复合动作包括顺序动作(EHSequence)、并行动作(EHFlow)和选择动作(EHSwitch)，根据异常处理逻辑的需要，可以扩展定义新的复合动作。

(1)顺序动作(EHSequence)

顺序动作(EHSequence)是将动作 $action_1$，$action_2$，…，$action_n$ 依次组合而成的复合动作，动作 $action_i$ 是原子动作或复合动作。其执行语义为如 EHSequence 的动作 $action_1$ 执行完成异常处理，则其后续动作被跳过，顺序动作活动完成执行；否则，继续执行 $action_2$，以此类推，直到异常被成功地处理或执行完 $action_n$。

$EHSequence := action_1 \triangleright action_2 \triangleright \cdots \triangleright action_n$

顺序动作是一种动作冗余容错机制的具体实现。动作 $action_1$，$action_2$ 满足定理 1 和定理 2，则 $action_1$ 和 $action_2$ 或 $action_2$ 和 $action_1$ 可顺序组合；如果动作 $action_1$ 和 $action_2$ 满足定理 3 和定理 4，则 $action_1$ 和 $action_2$ 可顺序组合，$action_2$ 和 $action_1$ 不能顺序组合。

(2)并行动作(EHFlow)

并行动作(EHFlow)是将动作 $action_1$，$action_2$，…，$action_n$ 并发组合而成的复合动作，动作 $action_i$ 是原子动作或复合动作。其执行语义为 $action_1$，$action_2$，…，$action_n$ 中任何一个处理动作执行成功，则 $EHFlow$ 动作完成，并终止其他未完成的异常处理动作。

$EHFlow := action_1 \parallel action_2 \parallel \cdots \parallel action_n, action_1, action_2, \cdots, action_n \in RA$

并行动作较顺序动作具有更好的性能，但以牺牲可信性能为代价。动作 $action_1$，$action_2$ 满足断言 1，则 $action_1$，$action_2$ 并行执行。

(3)选择动作(EHSwitch)

选择动作(EHSwitch)是根据 BPEL 流程运行时环境条件(condition)，选择异常处理行为的复合动作。

$EHSwitch = (condition, action_1, action_2)$

$condition$ 表示选择条件，$action_1$ 表示条件为真时的异常处理动作，$action_2$ 则表示条件为假时的异常处理动作。

3.3 异常处理过程

异常处理策略规范了 BPEL 流程的异常处理过程。BPEL 流程异常处理策略(EHPolicy)是异常处理规则(EHRule)的序列，$EHPolicy = \langle EHRule_1, EHRule_2, \cdots, EHRule_n \rangle$。异常处理策略目标 $Target = \{e_{bp1}, e_{bp2}, \cdots, e_{bpn}\}$ 定义了所能处理的异常类型集。当 BPEL 流程运行，抛出异常命中策略目标时，策略依次执行异常处理规则，直到规则执行成功或规则集执行完毕。返回方式(ReturnMode)表示异常处理路径返回到 BPEL 流程正常路径还是继续异常处理或终止异常处理。

3.4 异常处理动作转换方法

异常处理动作转换是保证 BPEH/PLD 策略能生成具有容错能力 BPEL 流程的关键技术。对原子动作转换，文献[3，11，12]都做了非常细致的工作，在此不再赘述。这里只给出复合动作的转换方法。

3.4.1 顺序动作

EHSequence 动作由多个异常处理动作按照先后执行的顺序组合而成。其转换算法描述如下：

1. 在故障作用域 s 中添加 Fault Handler，Compensate Handler 和 Terminate Handler；

2. 如为“anonymouseException”或“anyException”异常，则 Fault Handler 中增加〈catchAll〉活动，否则增加〈catch〉活动；

3. 在〈catchAll〉或〈catch〉活动中添加一个〈sequence〉活动；

4. 对于顺序动作定义的各个异常处理动作，按照其定义的顺序，采用前面定义的原子动作解析方法，将每个原子动作解析为对应的 WS-BPEL 代码；对于 Compensate 和 Cancel 动作，按照前面原子动作的转换规则，将在 Compensate Handler 和 Terminate Handler 中分别添加〈compensate〉或者〈terminate〉活动；

5. 如为复合动作，则按照复合动作的转换规则执行转换。

3.4.2 并行动作

EHFlow 动作定义了多个并发执行的异常处理动作，其转换算法描述如下：

1. 在故障作用域 s 中设置一个变量 $success$ 用于标识异常处理是否成功结束，默认为 false；

2. 在故障域 s 中添加 Fault Handler，Compensate Handler 和 Terminate Handler；

3. 如为“anonymouseException”或“anyException”异常，则在 Fault Handler 中添加〈catchAll〉活动，否则添加〈catch〉活动；

4. 在〈catchAll〉或〈catch〉活动中添加一个〈flow〉活动；

5. 将并行动作定义的各个原子动作转换为对应的 BPEL 代码；对于 Compensate 和 Cancel 动作，将在 Compensate Handler 和 Terminate Handler 中分别添加〈compensate〉或者〈terminate〉活动；

6. 如异常处理动作为复合动作，则按复合动作转换规则执行；

7. 如并发动作中某个异常处理动作成功完成异常处理，则设置前面定义的 $success$ 为 true；

8. 如 $success$ 为 true，则终止正在异常处理的其他异常动作。

3.4.3 选择动作

EHSwicth 动作定义了分支的异常处理动作，其转换算法描述如下：

1. 在故障作用域 s 中添加判断条件布尔变量 $condition$；

2. 在故障作用 s 中添加 FCT-Handlers；

3. 如为“anonymouseException”或“anyException”异常，在 Fault Handler 中添加〈catchAll〉活动，否则增加〈catch〉活动；

4. 在〈catchAll〉或〈catch〉活动中添加〈assign〉活动，条件赋值；

5. 在〈catchAll〉或〈catch〉活动中添加〈if〉活动；

6. 计算判断条件变量 $condition$，根据取值情况，分别执行对应异常处理动作；

7. 采用前面定义的原子动作解析方法，将每个原子动作

解析为对应的 WS-BPEL 代码；对于 Compensate 和 Cancel 动作，将在 Compensate Handler 和 Terminate Handler 中分别添加〈compensate〉或者〈terminate〉活动；

8. 如果异常处理动作为复合动作，则按照复合动作的转换规则执行转换。

4 BPEH/PDL 语言

为支持策略驱动的 BPEL 流程异常处理方法，在前面工作[13]的基础上，本文设计了一种新型的 BPEL 流程异常处理策略描述语言 BPEH/PDL(BPEL Process Exception Handling Policy Description Language)。BPEL 流程异常处理设计人员通过 BPEH/PDL 语言，声明式地定义 BPEL 流程异常处理策略，在较高抽象层次开发 BPEL 流程异常处理逻辑，实现了 BPEL 流程异常处理逻辑和正常业务逻辑的有效分离。

4.1 BPEH/PDL 的概念模型

BPEH/PDL 语言概念模型(见图 5)包括：异常处理策略集(EHPolicySet)、异常处理策略(EHPolicy)、异常处理规则(EHRule)和异常处理动作(EHAction)等核心元素。

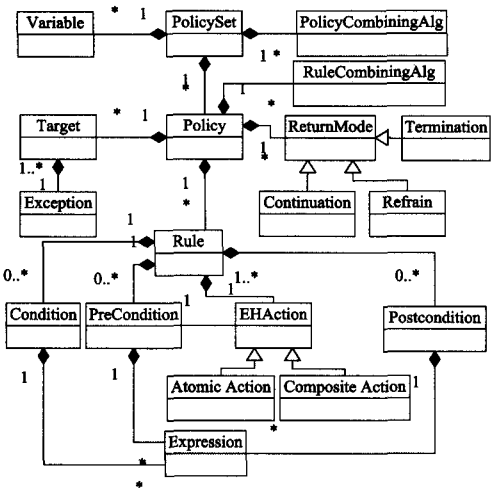

图 5 BPEH/PDL 语言元模型

策略集(EHPolicySet)是 BPEL 流程异常处理策略的集合，策略集的策略是相互独立且无序的。策略集在最高抽象层次描述了如何指导和控制 BPEL 流程异常处理的行为。策略变量定义 BPEL 流程异常处理时的流程状态和系统环境。策略组合算法元素(PolicyCombiningAlg)定义了策略集内策略冲突时的消解决策。策略(EHPolicy)是规则的集合。在 BPEH/PDL 语言中，策略处于策略集和规则的中间层，它既是策略集的基本构成单位，又是异常处理规则的容器。策略目标元素定义了策略所能处理的异常类型。策略是规则的顺序组合，即如前继规则执行失败，则执行后续规则，如此类推，直至执行成功或规则集执行完毕。如策略执行失败，则按照策略返回方式控制 BPEL 流程异常处理的行为。规则(EHRule)扩展了 ECA 规则范型，BPEH/PDL 规则定义了对 BPEL 流程异常的反应。

4.2 BPEH/PDL 的语法结构

BPEH/PDL 使用 XML 作为其元语言。BPEH/PDL 语言是面向 BPEL 流程异常处理的策略描述语言，既具有通用策略描述语言的基本元素和属性，又充分地考虑了 BPEL 流程异常处理策略描述的特点，支持从异常处理动作、规则、策略到策略集等不同抽象层次的描述。一个 BPEL 流程配置一个 BPEH/PDL 策略文件，策略文件至少包含一个 BPEL 流程异常处理策略集(EHPolicySet)，策略集是非空的 BPEL 流程异常处理策略(EHPolicy)的集合；策略是异常处理规则(EHRule)的非空集合，规则按照顺序组合执行；异常处理规则扩展了 ECA 规则范型，BPEL 流程异常发生时，如 BPEL 流程的状态满足规则使能的约束条件(Condition)和动作执行的前置条件(Precondtion)，那么执行规则的异常处理动作(EHAction)，并评估动作执行的效果。如策略所有规则执行失败，则按策略返回方式控制 BPEL 流程异常处理行为。BPEH/PDL 语言的总体语法结构如下：

```
〈xs:schema targetNamesapce=R2E〉
〈PolicySet PolicySetId=NCName PolicySetDefault=NCName?〉
〈Descriptions/〉?
〈Variables/〉?
〈PolicyCombingAlg/〉
〈Policy PolicyId=NCName Version=NCName? Priority=XS:Integer?〉
  〈Description/〉?
  〈RuleCombiningAlg/〉
    〈Target〉
      〈Exception〉+
    〈/Target〉
    〈Rule RuleId=NCName DateTime=XS:DateTime? Priority=XS:Integer?〉
      〈Condition/〉?
      〈PreCondition/〉?
      〈EHAction〉
      〈PostCondition〉?
    〈/Rule〉+
    〈ReturnMode/〉
  〈/Policy〉+
〈/PolicySet〉+
〈/xs:schema〉
```

5 BPEH/F 框架

为了支持基于 BPEH/PDL 策略的 BPEL 流程异常处理，实现 BPEL 流程异常处理逻辑和正常业务逻辑的分离，提高 BPEL 流程的可重用型和动态适应性，本文提出了基于 BPEH/PDL 策略的 BPEL 流程异常处理框架 BPEH/F。

5.1 BPEH/F 框架概述

软件框架是一种高效简洁的软件开发技术，提供软件实现复杂技术问题的概念结构，并为软件的具体实现和运行提供有效的支撑。BPEH/F 框架(见图 6)是一个基于 BPEH/PDL 语言、面向 Web 服务的 BPEL 流程异常处理框架。BPEH/F 框架为 BPEL 流程异常的检测、识别、传播和处理提供支持，为 BPEL 流程异常处理的设计和实现提供统一支撑环境。

BPEH/F 框架由设计时环境、运行时环境和服务资源集 3 部分组成。

(1)设计时环境(Development Environment)：为 BPEL 流程的异常逻辑开发人员提供基于 BPEH/PDL 策略的 BPEL 流程异常处理逻辑的规范、分析和部署功能。BPEL 流程异

常开发人员使用 BPEH/PDL 策略编辑器编写 BPEH/PDL 策略；为了保证 BPEH/PDL 策略的正确性，策略分析器在文献[14]工作的基础上，基于 BPEH/PDL 策略的形式化语义和 CPN Tools 仿真分析工具，支持 BPEH/PDL 策略关键属性(无冲突性、完备性、终止性、一致性等)的分析和验证[15]。分析和验证后的 BPEH/PDL 策略保存到 BPEL 流程异常处理策略库以备集成和调用。BPEL 流程的正常业务逻辑和 BPEH/PDL 策略在部署前，通过策略集成器生成具有容错能力的符合 BPEL 规范的 BPEL 流程，然后被部署到标准的 BPEL 引擎。

(2)运行时环境(Runtime Environment)：支持 BPEL 流程运行时的异常处理逻辑的动态配置和实现，包括 BPEL 流程异常检测模块、异常策略决策模块、异常策略执行模块和服务调用管理模块。

(3)服务资源集：框架提供 BPEL 流程异常处理服务和框架服务等服务资源集，支持对 BPEL 流程各类异常的处理。①系统异常的处理，提供对 BPEL 引擎、中间件以及操作系统功能的调用服务，如 BPEL 流程实例的挂起、恢复、终止等服务，以支持某些可修复的系统异常的处理。同时提供通知和日志框架服务，支持框架用户处理某些系统无法自动修复的异常。②资源异常的处理：资源异常是 BPEL 流程访问服务资源时发生了错误而引发的。某些资源异常可能是临时或随机原因造成的，如在服务提供方某时刻负载过大，造成服务访问超时，可以使用“重试”这一异常处理服务来处理该类异常。而另一些非临时性错误而引发的异常，如服务提供方不再提供某个服务，或者服务接口发生了变化，则可以采用“备用服务”、“自动接口适配后重试”或者“动态发现新的可用服务”等异常处理服务来处理。对于某些需要人工处理的资源异常，可以用日志和通知等框架内置的服务通知框架用户。③应用异常的处理：应用异常是服务或者流程的应用逻辑发生错误而引发的。该类异常是应用软件特定的异常。框架提供了扩展机制，支持开发人员为特定的应用异常、开发异常处理服务(也可使用 BPEL 流程语言来实现)，并在框架中注册，作为异常处理资源集新的组成部分。

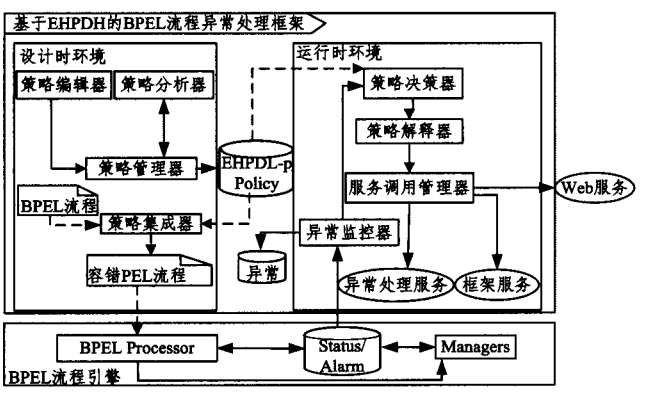

图 6 BPEH/F 框架

BPEH/F 框架的设计时环境、运行时环境和服务资源集构成了一个基于 BPEH/PDL、面向服务的、统一的 BPEL 流程异常处理框架。BPEH/F 框架既能满足 BPEL 流程部署阶段异常处理的需要，同时也满足 BPEL 流程运行时异常处理的动态配置要求。通过注册新的异常处理服务资源和框架服务资源，支持 BPEH/F 框架功能的扩展。

5.2 BPEH/F 框架核心功能

5.2.1 策略编辑器

策略编辑器为 BPEL 流程异常设计人员提供了一个图形化的集成设计环境，支持基于 BPEH/PDL 语言的 BPEL 流程异常处理策略的开发。策略编辑器是一个可视化的基于 BPEH/PDL 语言的策略开发工具。策略编辑器总体上包括 3 大功能：BPEH/PDL 策略浏览、BPEH/PDL 策略编辑、BPEH/PDL 策略语法检查。BPEL 流程异常处理逻辑开发人员可视化地定义 BPEH/PDL 策略的各个构成元素和设置元素属性，高效灵活地完成基于 BPEH/PDL 的 BPEL 流程异常处理逻辑的开发。策略编辑模块以 Eclipse 插件的形式存在，既可被集成到 BPEL 流程的集成开发环境中，也可以作为一个独立的软件使用。

5.2.2 策略分析器

策略分析器基于 BPEH/PDL 语言的形式化语义模型，通过调用 CPN Tools 的仿真分析工具，支持 BPEH/PDL 策略冲突的检测和消解以及规则冲突的检测和消解。策略分析器是一个图形化的 BPEH/PDL 策略分析工具，是 BPEH/F 框架和 CPN Tools 的接口。策略分析器总体上包括两项功能：BPEH/PDL 策略的冲突检测和消解、规则的冲突检测和消解。

(1)BPEH/PDL 策略的冲突检测和消解

框架为策略设计人员提供 BPEH/PDL 策略冲突的检测和消解服务。根据策略冲突的检测和消解过程，BPEH/PDL 策略的冲突检测和消解细分为 BPEH/PDL 策略转换、BPEH/PDL 策略冲突检测和 BPEH/PDL 策略消解 3 个子功能。

①BPEH/PDL 策略转换：基于 BPEH/PDL 语言的 CPN 模型，将 BPEH/PDL 策略集和策略集内的策略转化为对应的 CPN 模型；策略集和策略的 CPN 模型将作为 CPN Tools 仿真分析工具的输入。

②BPEH/PDL 策略冲突检测：利用 CPN Tools 仿真分析工具，以转换后的策略集和策略 CPN 模型为输入信息，调用策略集 CPN 模型的冲突检测模块实现策略冲突的检测。

③BPEH/PDL 策略冲突消解：如果冲突检测模块已检测到策略集中存在着策略冲突，则调用策略消解模块实施策略冲突的消解。

(2)BPEH/PDL 规则的冲突检测和消解

BPEH/PDL 规则冲突检测和消解基本思路与策略冲突的检测和消解相似，包括 BPEH/PDL 规则转换、BPEH/PDL 规则冲突检测和 BPEH/PDL 规则冲突消解 3 个子功能。

5.2.3 BPEH/PDL 策略集成器

策略集成器是 BPEH/F 框架中实现部署阶段 BPEL 流程异常处理的关键组件，它为 BPEL 流程异常管理人员提供了可视化的 BPEL 流程正常业务逻辑和基于 BPEH/PDL 策略的异常处理逻辑的集成功能。策略集成器包括 3 项功能：BPEH/PDL 策略配置、策略转换和策略植入。BPEL 流程异常管理人员可灵活地为 BPEL 流程正常业务逻辑配置 BPEH/PDL 策略，框架自动地将 BPEH/PDL 策略转换为标准的 BPEL 程序，最后通过策略植入模块，生成具有容错能力

的符合 WS-BPEL 规范的 BPEL 流程被部署到标准的 BPEL 流程引擎(如 ActiveBPEL)。

(1)策略配置

框架为 BPEL 流程管理人员或开发人员提供了可视化的策略配置功能。策略配置细分为活动配置和全局配置。活动配置是将 BPEH/PDL 策略挂接到 BPEL 流程的 Invoke 活动和 Scope 活动,每个活动只能配置一个 BPEH/PDL 策略。全局配置是为整个 BPEL 流程模型配置一组 BPEH/PDL 策略,策略组构成 BPEL/PDL 策略集。

(2)策略精化

BPEH/PDL 策略需要精化为 WS-BPEL 代码后才能被 BPEL 流程引擎执行。BPEH/PDL 策略转换功能为 BPEL 流程异常管理和开发人员提供了自动化的 BPEH/PDL 策略的精化功能,可自动地将表示 BPEL 流程异常处理行为的 BPEH/PDL 策略转化为符合 WS-BPEL 规范的 BPEL 代码。策略精化包括策略集精化、策略精化、规则精化和动作精化。动作精化是 BPEH/PDL 策略转换的基础和前提,其转换实现过程见 3.4 节。规则精化包括条件转换和动作转换,规则条件可以转换为 WS-BPEL 的 If 活动;策略转换包括策略目标转换、规则转换和返回方式转换。策略目标可转换为 WS-BPEL 故障处理器的 Cacth 活动集合,策略返回方式的抑制模式、继续模式和终止模式可分别转换为活动或流程的 Suppression 属性设置、Rethrow 活动和 Exit 活动。策略集转换就是策略转换的遍历过程。

(3)策略植入

框架通过调用 BPEH/PDL 策略转换服务,自动将已经完成 BPEH/PDL 策略转换的 WS-BPEL 代码植入到活动或流程相应的故障处理器中,生成具有容错能力的 BPEL 流程。

5.2.4 策略决策器

策略决策器是 BPEH/F 框架中实现运行时异常处理的核心组件,为 BPEL 流程提供运行时异常处理的决策功能。策略决策器功能包括:异常处理请求格式转换、BPEH/PDL 策略检索和策略选择。

(1)异常处理请求格式转换

异常监控器从 BPEL 流程引擎的队列管理器实时地获取 BPEL 流程运行时的状态和异常信息。当 BPEL 流程运行发生异常时,策略决策器首先接收异常监测器发送的异常处理请求,并将异常处理请求转化为符合策略执行器要求的异常处理请求格式。

(2)策略检索

策略决策器根据格式转换后的异常处理请求信息,检索 BPEH/PDL 策略库,返回结果为满足处理当前异常处理请求的策略列表。根据返回的策略列表,进行不同的策略决策。

(3)策略选择

策略检索返回的策略列表存在着多种可能性。如列表不为空,则表示存在能够满足当前异常处理请求的一条或多条 BPEH/PDL 策略。如只有一条 BPEH/PDL 策略,决策器将该策略传递给策略执行器执行;如有多条异常处理策略,则根据策略优先级选择其中一条执行。如不存在满足当前异常处理请求的策略,则暂停 BPEL 流程运行,提示 BPEL 流程管理人员新增或重新配置 BPEH/PDL 异常处理策略,完成配置后再重新执行前面的决策过程。

5.2.5 策略执行器

策略执行器为 BPEL 流程提供自动运行时异常处理功能。策略执行器本身并不完成具体的 BPEL 流程异常处理行为,它是策略决策器和服务调用管理器的中介,当接收到策略决策器发送来的 BPEH/PDL 策略时,执行器首先解析待执行的 BPEH/PDL 策略,生成策略的规则队列;遍历策略的规则队列,解析每条策略规则,生成待执行规则动作;解析待执行规则动作,将动作映射到相应异常处理服务或框架服务,将服务请求信息传递给服务调用管理器,由服务调用管理器负责调用相关的服务资源。策略执行器主要包括 BPEH/PDL 策略解析、规则解析和动作映射 3 个功能。

(1)策略解析

策略解析是将策略决策器输出的 BPEH/PDL 策略转换为 BPEH/PDL 执行队列的过程。策略解析模块首先读取策略决策器输出的 BPEH/PDL 策略,遍历 BPEH/PDL 策略内嵌的规则,生成待执行的 BPEH/PDL 规则队列;然后,按照规则队列中规则的优先级定义,对规则队列的规则重新排序,生成优化的待执行 BPEH/PDL 规则队列,交给 BPEH/PDL 规则解析模块进一步解析。

(2)规则解析

规则解析模块遍历优化 BPEH/PDL 规则队列。首先,解析异常处理请求得到 BPEL 流程异常发生时的上下文信息,计算规则使能条件是否为"True",如规则使能条件满足,则输出规则定义异常处理动作到 BPEH/PDL 动作解析模块,并从规则队列中删除。如规则使能条件为"False",则从队列中删除该规则,继续后继规则的解析,直至规则队列为空。

(3)动作解析

动作解析是将规则异常处理动作映射异常处理服务或框架服务的过程。BPEH/PDL 动作只是在较高抽象层次描述了 BPEL 流程异常处理行为,而实际的 BPEL 流程异常处理由框架注册的异常处理服务或框架内嵌服务(如日志服务、通知服务等)实现。对于原子动作而言,其对应着某个异常处理服务或框架服务;而复合动作则解析为多个异常处理服务或框架服务的服务组合。

5.2.6 服务调用管理器

服务调用管理器负责实施 BPEL 流程运行时异常处理行为。服务调用管理器支持对异常处理服务资源和框架服务资源的服务注册和服务调用功能。服务注册功能将新的异常处理服务或框架服务注册到 BPEH/PDL 框架,扩展了 BPEH/PDL 框架的异常处理能力。服务调用功能负责对已注册的异常处理服务资源或框架服务资源的调用。服务注册和服务调用功能非常明确,在此不再展开描述。

5.3 BPEH/PDL 策略实施机制

在 BPEL 流程设计过程中考虑到所有的异常及其处理是不现实的。因此,需要在 BPEL 流程执行时提供运行时异常处理机制,以进一步提高 BPEL 流程可信性。BPEL 流程运行时异常处理机制涉及到两个关键方面:(1)如何根据 BPEL 流程发生异常时的异常信息和上下文信息选择正确的处理策略;(2)如何正确地执行已决策的异常处理策略。

(1)策略决策机制

策略决策机制采用发布-订阅模式,当异常监控器检测

(下转第 192 页)

cy. 2006:32-46

[18] Wang W, Murynets I. What you see predicts what you get lightweight agent based malware detection[J]. Security and Communication Networks, 2012, 6(1): 33-48

[19] Geer D. Behavior Based Network Security Goes Mal-stream[J]. Computer, 2006, 39(3): 14－17

[20] Bayer U, Moser A, Kruegel C, et al. TAnalyze: Dynamic Analysis of Malicious Code[J]. Journal in Computer Virology, 2006, 2(1): 67-77

[21] Harmer P K, Williams P D, Gunsch G H, et al. Artificial Immune System against Viral Attack[J]. IEEE Transactions on Evolutionary Computation, 2002: 353-359

(上接第 186 页)

BPEL 流程引擎异常事件时，异常监控器通知异常决策器。决策器接收异常处理请求 $ehreq_0$，请求预处理器将 $ehreq_0$ 格式转换为策略执行器要求的格式 $ehreq_1$。策略检索器以 $ehreq_1$ 为查询参数，从策略库中检索满足 $ehreq_1$ 的策略列表 *policyList*。如果策略列表为空，则调用策略编辑器新增策略或修改已有策略。如果策略长度为 1，则只存在一条满足当前异常处理请求的策略，直接输出该策略到策略执行请求。如果策略长度大于 1，则存在多条满足当前异常处理请求的策略，按照策略选择设置，①先进先出(FIFO)：输出策略列表的第一个元素；②后进先出(LIFO)：输出策略队列的最后元素；③随机选择：随机从策略列表中选择一条策略；④优先选择：从策略队列中选择优先级最高的策略到策略执行请求。

(2)策略执行机制

BPEH/PDL 策略是 BPEL 流程异常处理逻辑的抽象表达，它是一套指导和决定如何管理和控制 BPEL 流程异常行为的相对持久的、说明性的规则序列。策略执行机制是 BPEH/F 框架中策略决策器和服务管理器的中介，将策略决策器输出的较高抽象层次的 BPEH/PDL 策略转化映射为框架服务或异常处理服务调用请求。

1. 策略解析器接收策略决策器输出的策略执行请求 *policyExecReq*，生成策略规则队列 $ruleQueue_0$；

2. 规则队列管理器对生成的规则队列 $ruleQueue_0$ 排序，生成具有优先执行顺序的规则队列 $ruleQueue_1$；

3. 规则遍历器遍历规则队列 $ruleQueue_1$，并调用规则解析器对规则进行解析；

4. 规则解析器解析 EHPDL-P 规则，调用规则评估器计算规则使能条件；

5. 规则评估器根据策略执行请求，计算规则使能条件是否满足，如果满足使能条件，则返回逻辑真值，否则，返回逻辑假；

6. 规则解析器接受规则评估器返回的评估结果为真值时，将规则动作作为映射请求发送给动作映射器；

7. 动作映射器将规则动作映射为框架服务或异常处理服务请求。

结束语 本文在已有研究工作基础上，首先分析了策略驱动的 BPEL 流程异常处理机制，设计了一种新的 BPEL 流程异常处理策略描述语言 BPEH/PDL，提出了一种基于 BPEH/PDL 的集成化 BPEL 流程异常处理框架 BPEH/F。下一步的研究工作将进一步完善 BPEL/F 框架功能，开发基于 BPEH/PDL 的 BPEL 流程异常处理支撑工具，提高 BPEH/F 框架的异常处理能力。

参 考 文 献

[1] Curbera F, Khalaf R, Leymann F. Exception Handling in the BPEL4WS Language[C]// BPM 2003. LNCS 2678, 2003: 276-290

[2] Boutaba R, Aib I. Policy-Based Management: A Historical Perspective[J]. Journal of Network and Systems Management, 2007, 15(4): 447-480

[3] Zeng Liang-zhao, Lei Hui, Jeng Jun-jang. Policy-Driven Exception-Management for composite Web services[C]// Conference on Electronic Commerce－CEC. Florence, Italy. 2005: 355-363

[4] Charfi A, Mezini M. AO4BPEL: An Aspect- oriented Extension to BPEL[J]. World Wide Web, 2007, 10: 309-344

[5] Liu An, Li Qing, Huang Liu-sheng, et al. FACTS: A Framework for Fault-Tolerant Composition of Transactional Web services [J]. IEEE Transactions on Service Computing, 2010, 3(1): 46-59

[6] Erradi A, Maheshwari P. AdaptiveBPEL: a Policy-Driven Middleware for Flexible Web Services Compositions[C]// Proceedings of Middleware for Web Services(MWS). 2005: 5-12

[7] Erradi A, Maheshwari P. Policy-driven middleware for self-adaptation of Web services compositions[C]// IFIP International Federation for Information Processing 2006. Middleware 2006, LNCS 4290, 2006: 62-80

[8] Erradi A, Tosic V, Maheshwari P. MASC-. NET-Based Middleware for Adaptive Composite Web Services[C]// International Conference on Web Services(ICWS 2007). 2007

[9] Karastoyanova D, Leymann F. BPEL 'n' Aspects: Adapting Service Orchestration Logic[C]//2009 IEEE International Conference on Web Services. 2009: 222-229

[10] Karastoyanova D, Khalaf R, Schroth R, et al. BPEL Event Model [R]. University Stuttgart, 2006

[11] 刘安. Web 服务驱动的业务流程的容错性研究[D]. 合肥：中国科学技术大学，2008

[12] 刘海，刘安，李青，等. 一种 ECA 规则驱动的 BPEL 流程异常处理和分析机制[J]. 小型微型计算机系统，2010，31(7)：1363-1370

[13] 王权于，应时，吕国斌，等. 一种面向服务流程异常处理的策略描述语言[J]. 计算机科学，2012，39(2)：148-153

[14] 蒋曹清，应时，文静，等. 面向服务软件中异常处理的形式化建模方法[J]. 西安交通大学学报，2013，47(4)：118-124

[15] 蒋曹清，应时，文静，等. 面向服务软件异常处理过程的可终止性验证[J]. 计算机科学与探索，2012(3)：208-220