一种基于CPN的BPEL异常处理逻辑的开发方法

管 华[1,2] 应 时[1] 贾向阳[1] 蒋曹清[1] 王一兵[3]
(武汉大学软件工程国家重点实验室 武汉 430072)[1] (武汉工业学院网络中心 武汉 430023)[2]
(第二炮兵指挥学院三系 武汉 430012)[3]

摘 要 针对WS-BPEL在面向服务软件异常处理方面不够完善的问题,提出了一种基于着色Petri网描述的BPEL异常处理逻辑开发方法。该方法利用着色Petri网(CPN)形式化地描述BPEL的异常处理机制,建立BPEL的异常处理CPN模型,指导对BPEL的异常处理逻辑开发,并依据此建模思想,提出了一个BPEL的异常处理CPN模型的转换工具,用以实现将异常处理的BPEL着色Petri网模型转换成对应的带异常处理的BPEL代码。该工具可在输入的原始的没有异常处理功能的BPEL代码基础上,通过动态地添加BPEL的异常处理语言成份,形成带有异常处理功能的BPEL流程。最后通过引入供应商流程案例,阐述了该方法的具体使用过程。

关键词 着色Petri网,BPEL(Business Process Execution Language),异常处理
中图法分类号 TP311 **文献标识码** A

Development Approach of Exception Handling Logic for BPEL Process Based on Coloured Petri Net

GUAN Hua[1,2] YING Shi[1] JIA Xiang-yang[1] JIANG Cao-qing[1] WANG Yi-bing[3]
(The State Key Lab of Software Engineering, Wuhan University, Wuhan 430072, China)[1]
(Network Center, Wuhan Polytechnic University, Wuhan 430023, China)[2]
(No. 3 Dept., The Second Artillery Command Coll, Wuhan 430012, China)[3]

Abstract According to the problem of exception handling for WS-BPEL in service oriented software, this paper presented an approach for BPEL's exception handling based on color petri net(CPN). Through formalizing description of exception handling mechanism of BPEL, the approach builds exception handling CPN model, instructs the development of BPEL's exception handling logic, and based on this modeling idea, gives a converting tool for the BPEL's exception handling CPN model, which converts the BPEL's exception handling CPN model into BPEL codes with exception handling function, and through danamically adding exception handling elements of BPEL, the tool can produces new BPEl process with exception handling. Finally we represented a case study to illustrate the using process and prove the affectivity of our method.

Keywords Color Petri net, BPEL(Business Process Execution Language), Exception handling

1 引言

WS-BPEL(Web Service Business Process Execution Language,BPEL)即业务流程执行语言,是基于业务流程建模的规范而产生的语言。BPEL能够把几种完全不一样的Web服务通过相关的需求业务逻辑紧密地衔接起来,创建一个更加庞大并且完善的业务流程[1]。BPEL的出现解决了单个Web服务功能有限的问题,通过集成多个Web服务的功能,提供更加完善的服务功能,可以提高系统的运行效率,快速地满足用户的需求。然而,面向服务软件由于运行环境具有动态性和不确定性,服务资源具有自治性和松耦合性,这些都使得面向服务软件在运行过程中会面临许多新的且更复杂的异常情况,设计出的服务流程都有可能发生设计故障的问题,或者由于技术人员的开发失误产生的流程与实际业务冲突的问题,而这些问题往往在开发的服务组合业务流程完成后,甚至到实际运行时才可能被发现。目前,BPEL提供了一定的异常处理机制来显式地捕获这些错误,如通过引入faulthandler元素封装相应的异常处理子程序。此外,已完成的活动在晚些时候可能需要被撤销,因为这些活动是一个时间更长的事务的一部分,而这个事务不得不被终止。BPEL也引入了compensationhandler元素,以用于定义在发生异常时或伙伴请求撤销时流程中单个或合成活动是怎样被补偿的。

到稿日期:2012-03-12 返修日期:2012-07-26 本文受国家自然科学基金项目(61070012,61070022),国家自然科学基金重点项目(91118003)资助。

管 华(1978—),男,博士,工程师,CCF会员,主要研究领域为面向服务软件工程、Petri网应用,E-mail:11850075@qq.com;**应 时**(1965—),男,教授,博士生导师,CCF高级会员,主要研究领域为面向对象软件工程方法、软件体系结构和模式、软件的可重用性与互操作性;**贾向阳**(1972—),男,博士,讲师,主要研究领域为面向服务的软件开发、面向服务的中间件;**蒋曹清**(1973—),男,博士生,讲师,主要研究领域为petri网、软件体系结构。

尽管 BPEL 中提供了类似程序设计语言的异常处理成分(这是和传统工作流的一个显著区别),但 BPEL 的异常处理机制仍存在着一些不足:异常流和正常流程紧密耦合;异常处理描述能力不足;流程的可读性较差;需要流程设计人员全面考虑各种类型的异常;服务发生变化时如使用新服务替换原服务,新服务可能抛出新异常,需要对流程进行修改;异常发生时,正常流程直接终止后续的执行;没有提供 retry、resume 等机制。为了能够让建立起来的业务流程更加安全高效地运行起来,应该在流程设计阶段使用形式化方法对 BPEL 流程的异常处理进行建模,及时发现 BPEL 流程中的异常,分离正常业务和异常流程,从而保证业务流程的稳定运行。本文围绕上述问题提出一种 BPEL 流程的异常处理方法,该方法利用着色 Petri 网(CPN)进行形式化建模。

CPN 是一种图形化语言,可用于建模和验证并发系统、分布式系统和并发起重要作用的系,它首先由丹麦奥尔胡斯大学的 Kurt Jensen 在博士论文中提出[2]。CPN 是一种整合 Petri 网能力和高级编程语言能力的离散事件建模语言。Petri 网提供图形符号和对并发、通信和同步建模的基本原语的基础。CPN 可以通过交互或自动地进行仿真,通过状态空间进行相关性质分析,从而可进一步分析所建模型的合理性和逻辑正确性。

本文首先介绍 BPEL 异常处理建模的相关研究工作,然后提出了一种 BPEL 流程的异常处理逻辑开发方法,并阐述了该方法的基本思想;接着将 BPEL 的几个跟异常处理相关的活动(包括 Scope、异常处理、补偿处理、终止处理)分别映射成着色 Petri 网模型,介绍了一个 BPEL 的异常处理 CPN 模型的转换工具;最后以一个供应商流程案例阐述了如何应用本文提出的 BPEL 异常处理逻辑开发方法。

2 相关研究

目前,已经提出很多方法可建模 BPEL 异常流程处理,如北京大学的裘宗燕等基于 CSP 的 PPL 语言、Web 服务编排中的协同异常处理过程[3]。英国玛丽皇后伦敦大学的 Marco Carbone 提出了 Web 服务编排时交互异常的处理方法[4],其通过协同多个参与交互的 Web 服务来处理交互过程中产生的异常,并且基于 global 演算给出了该异常处理机制的形式化语义。加拿大国家研究理事会的 Yuhong Yan 等人提出了一种 Web 服务编制的建模和诊断方法[5],该方法将 BPEL 流程转换成同步自动机模型,当异常抛出时,基于该形式化模型计算流程执行的轨迹,并通过轨迹上的变量依赖诊断异常的发生。其它还有用 PI 演算、时序逻辑、B 方法等来建模 Web 服务组合过程中的异常处理问题。

本文主要关注 Petri 网对 BPEL 异常处理流程的建模。目前有很多文献提出了将 BPEL 转化为 Petri 网的语义规则。最早提出将 BPEL 过程映射到 Petri 网的是荷兰科学家 W. M. P. Van Der Aalst[6],他提出将 BPEL 过程映射成 WF-nets (Petri 网的一种),从而利用其完备的理论体系和丰富的工具来完成对 BPEL 过程各方面性质的验证。在此之后,有很多的研究人员提出自己的 BPEL 到 Petri 网的映射方式,澳大利亚昆士兰技术大学的 Chun Ouyang 提出一个综合的和严格的定义 BPEL 控制流结构、链接、事件和异常处理到 Petri 网结构的映射[7]。其使用"skip"行为代表不能被执行的活动,把每个基本活动分为建模初始状态、最终状态、已经开始、结束状态 4 个状态。在 SCOPE 里定义了 4 个标志映射异常处理,而并提出可基于现有 Petri 网分析技术,把从 BPEL 到 Petri 网的映射结果用来执行形式化的校验和分析 BPEL 过程。

德国洪堡大学 Hinz S 等提出了 BPEL 的一个完全特征的 Petri 网语义[8]。其 Petri 网语义由模式组成,每一个 BPEL 结构存在一个模式,通过 stop 组件来扩展每一个活动的模式和 SCOPE 模式。柏林洪堡大学和 Chun Ouyan 的主要问题是结构比较复杂,考虑情形较多,缺乏实验的手段。柏林洪堡大学 Schmidt 也提出另一个基于 Petri 网的 BPEL 语义[9],它能建模 BPEL 的所有特征。数据和相关集建模为对象在变迁中使用,跟活动相关的链接必须包括在链接的行为里。此外,澳大利亚新南威尔士大学的 Rachid Hamadi通过扩展 Petri 网提出了一种自适应恢复网(SARN)[10],SARN 是一种基于扩展 PETRI 网的 BPEL 流程恢复策略描述语言。SARN 通过恢复变迁和令牌来建模 BPEL 流程的异常处理。活动发生故障时,引发异常事件,恢复变迁激活,执行与之相应的操作序列(如创建库所和删除弧)来处理引发的异常。

值得一提的是,国内学者在这方面也有一些研究成果,华东理工大学的虞慧群、范贵生等人利用 Petri 网对服务组合中不同的故障情况进行建模[11],并依据组件的关系生成服务组合的故障处理模型。西安电子科技大学的段振华教授[12]、同济大学的孙健博士[13]等人同样基于 Petri 网对 BPEL 的活动进行了映射,以验证 Web 服务组合。

上述研究尽管取得了一定的成果,但大部分工作主要是建立在基本的 Petri 网的理论上。基本 Petri 网在对 BPEL 的异常处理建模时,存在一些不足,如在原有的正常 Petri 网模型基础上,需要增加更多库所表示异常处理,容易产生状态空间爆炸问题;不能清晰描述异常处理行为的一些属性,特别是在正常流程到异常流程的跳转上面,不能清晰表达跳转动作执行的条件;并且在 BPEL 的异常处理建模方面做得还不够完善,不能全面地反映 BPEL 的异常处理特性,也未提及建模后的应用实现等问题。因此本文提出了一种基于着色 Petri 网的 BPEL 流程的异常处理建模方法,以完善 BPEL 的异常处理机制。

3 BPEL 流程的异常处理逻辑开发方法

3.1 基本思想

BPEL 流程的异常处理逻辑开发方法的基本思想分为 3 步,第一步利用着色 Petri 网形式化地描述 BPEL 的异常处理逻辑开发方法,该方法把 BPEL 的每个活动建模为 CPN 模型,将 BPEL 代码的每个活动用 CPN 建模;第二步在本文提出的面向服务的异常处理转换工具的基础上,将带有异常处理的着色 Petri 网模型转换成对应的具有异常处理的 BPEL 代码;第三步是对第二步的有异常处理的 BPEL 代码进一步解析,将其转换成标准的、完善的异常处理的 BPEL 代码。

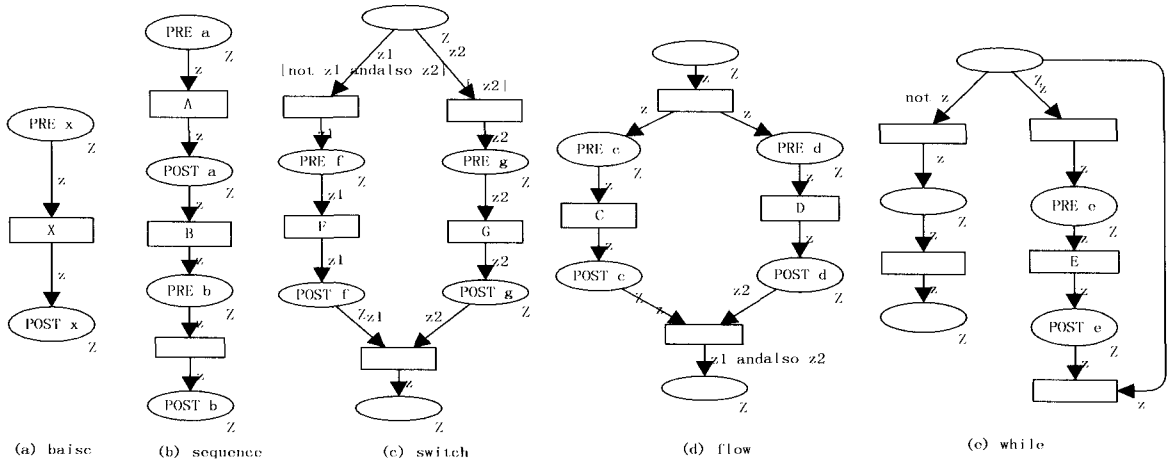
下面介绍第一步做法的基本思路:

(1) 将 BPEL 中的每个活动建模为一个变迁,在每个可能发生异常的变迁(活动)之前添加一个前置条件(precondition)库所,变迁之后添加一个后置条件(Postcondition)库所。

如图 1(a)所示，一个 BPEL 的活动 x 建模为一个变迁 X。PRE_x 是 x 的前置条件库所，当准备开始活动 x 前，它检查 x 是否存在异常；POST_x 是 X 的后置条件，标示 x 的异常处理是否正常结束。基于此方法，可建模其它 BPEL 活动，图 1(b)是 BPEL 的〈sequence〉活动的异常处理 CPN 模型，它有 A，B 两个变迁(BPEL 活动)，PRE_a，POST_a 是 A 的前置条件和后置条件，PRE_b，POST_b 是 B 的前置条件和后置条件。图 1(c)是 BPEL 的〈swich〉活动的异常处理 CPN 模型，它有 F，G 两个并发变迁(BPEL 活动)，PRE_f，POST_f 是 F 的前置条件和后置条件，PRE_g，POST_g 是 G 的前置条件和后置条件，BPEL 的〈pick〉活动的 CPN 模型跟图 1(c)的〈swich〉活动的 CPN 模型类似，差别只是图 1(c)的 PRE_f，PRE_g 前面的分支选择变迁的卫士条件表达式不同，因此本文省略了〈pick〉活动的 CPN 模型。图 1(d)是 BPEL 的 flow 活动的异常处理 CPN 模型，它有 C，D 两个并发变迁(BPEL 活动)，PRE_c，POST_c 是 C 的前置条件和后置条件，PRE_d，POST_d 是 D 的前置条件和后置条件。图 1(e)是 BPEL 的〈while〉活动的异常处理 CPN 模型，循环体内的变迁是 E，循环体外的变迁是 K，PRE_e，POST_e 是 E 的前置条件和后置条件，PRE_k，POST_k 是 K 的前置条件和后置条件。

(2) 在 BPEL 的 Scope 中，定义了 2 个库所 RUN 和 STOP。其中，STOP 库所标志一个错误发生，停止 Scope 里正常的活动；RUN 库所表示错误正常结束，恢复 Scope 里活动正常执行。第 3.2—3.5 节将详细介绍如何把 BPEL 的异常处理语言成份转换成带异常处理的 CPN 模型。

第二步的基本思路是在第一步的基于着色 Petri 网的 BPEL 的异常处理模型上，基于第 4 节提出的转换工具，在没有异常处理或异常处理不完善的原始 BPEL 代码上增加一些跟该模型对应的异常处理相关的 BPEL 代码，具体需要添加的异常处理相关的 BPEL 代码的原则如下。



(a)基本活动的异常处理 CPN 模型 (b)sequence 的异常处理 CPN 模型 (c)switch 的 CPN 模型 (d)flow 的 CPN 模型 (e) while 的异常处理 CPN 模型

图 1

(1) 首先需要在 BPEL 代码中可能发生异常的活动(如〈invoke〉，〈receive〉，〈reply〉等)之前添加一个 pre-condition 服务的调用，在活动之后添加一个 post-condition 服务的调用。pre-condition 服务判断该活动会不会产生异常，如果有异常，就将正常执行转到异常处理程序，同时中止正常的活动。post-condition 服务判断该活动的异常处理是否正常结束，如果正常结束，就恢复正常活动的执行。

(2) 在 scope 里设置 stop 和 run 两个相关集，初始时，这两个相关集的值都为假。stop 相关集在异常处理之前设置，一旦设置该相关集的值为真，则该 scope 中，除正在进行的异常处理活动能执行外，所有其它的活动都要终止，直到 run 相关集的值为真。当正在进行的异常处理活动正常结束时，run 相关集的值为 true，同时继续执行 scope 中其它的活动。

第三步是把第二步有异常处理的 BPEL 代码进一步解析成标准的有异常处理功能的 BPEL 代码。主要是通过异常处理转换工具，并通过匹配转换工具中的异常处理策略中定义的 BPEL 代码转换规则，将第二步添加的代码(如 pre-condition 服务调用，post-condition 服务调用)执行后转换成对应的标准 BPEL 代码。具体的转换规则可能有许多，下面只简单介绍几个主要规则。

(1) 当 pre-condition 服务检测到有异常发生时，可以在 pre-condition 服务调用的执行中插入〈faultHandlers〉，cathall 等异常处理相关的活动，在 post-condition 服务调用执行时，对应地插入〈faultHandlers〉，cathall 等活动的结束标志。

(2) 当异常发生时，要跳过一些活动的执行，可以在 pre-condition 服务调用的执行中插入一个〈scope〉活动的开始标志，且把 exitOnStandardFault 的值设为“yes”。exitOnStandardFault为“yes”时，如果〈scope〉中产生了异常，自动终止 scope 中后续活动的执行。exitOnStandardFault 为“no”时，需要〈scope〉自己来处理异常。在 post-condition 服务调用执行时，在 run 相关集值为真的地方插入一个〈scope〉活动的结束标志。

(3) 如要实现 BPEL 的 retry、resume 等机制，可以在 pre-condition 服务调用，post-condition 服务调用执行时，插入 while，if 等控制成分。

本文第 4 节具体介绍上面提到的面向服务的异常处理转换工具如何实现上述的第二步和第三步的思路，并简单介绍该转换工具的主要组成。

3.2 Scopes 的异常处理 CPN 模型

Scopes 提供了一个环境去影响它封闭活动执行的行为，它可以被任意深度地嵌套和结构化。scopes 为嵌套在其中的活动提供上下文，并且也是定义故障处理程序和补偿处理程序的地方。图 2 建模 scopes 中的异常处理，为有助于映射异常处理，我们添加了 2 个库所：STOP 和 RUN，STOP 库所表

示一个错误发生，scopes 里的所有涉及的活动都要停止，它在异常处理前获得一个令牌。RUN 库所表示 scopes 里的活动继续正常执行，在异常处理完后且 STOP 库所中有令牌时，它才能获得令牌。添加 STOP 和 RUN 库所的主要目的是记录异常发生的起始点和结束位置，以便从正常流程切换到异常处理流程。

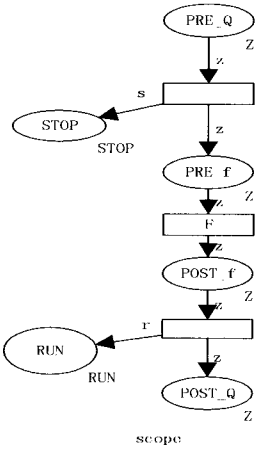


图 2　scope 活动的 CPN 模型

3.3　Fault handlers 的异常处理 CPN 模型

Fault handlers 是发生问题时的故障处理器，在 BPEL 中被看作是正常进程的一个 switch 模式。故障多半出现在接收故障信息或者对〈throw〉活动进行明确调用的时候。Fault handlers 在一个 scopes 中定义，指定在 scopes 执行中可能发生错误的应对行为，使过程从本地可能预见的错误执行过程中得以恢复。当一个错误码在一个 scopes 的正常过程中发生时，错误码将被一个在 scopes 中定义的〈faulthandlers〉活动捕获，同时，scopes 从正常过程模型切换到异常处理模型。

一个 Fault handler 由〈catch〉语句、〈catchall〉语句组成，通常指定几个〈catch〉活动为特定的错误，指定〈catchall〉活动为其它错误码。每个活动里，错误能被捕获且能被处理、指出。Fault Handlers 的〈catch〉活动可以在 3 个地方定义：在〈scope〉中定义、在〈process〉中定义（全局的 scope）、直接在〈invoke〉中定义。每个〈catch〉的属性有“faultName”或者“faultVariable”结构被定位拦截一个具体的故障。〈throw〉在正常流中使用，将流程中自定义的异常抛出去；〈rethrow〉在〈catch〉中使用，将捕获的异常重新抛出去。

在图 2 的 scope 中，在一个活动 X 执行前有一个异常处理（fault handlers），抛出异常后，流程转向异常处理程序。通过传令牌给 stop 库所，隐式地终止直接包括在其中的所有正在执行的活动。每个基本活动或正常事件需要检查现存的 run 库所的令牌，如果一个布尔值为 false 的令牌在 stop 库所里提供，它将能正常执行，否则不能执行。这应用到所有基本活动和整个过程的正常事件。而异常处理程序在执行完毕之后，既可以将异常继续抛给上层 scopes，也可以转向 scopes 的运行变迁使流程继续运行。图 3 中的 CPN 模型将 BPEL 的正常流程和异常流程清晰地分离出来，其中，右边是正常流程，左边是异常处理流程，hf 变迁是〈faulthandlers〉活动，PRE_hf、POST_hf 分别是 hf 的前置条件和后置条件；左边正常处理程序在调用〈faulthandlers〉活动前给一个令牌到 STOP 库所，hf 变迁执行完后，当 STOP 库所和 POST_hf 都有令牌时，run 库所获得令牌，接着从异常处理程序切换到正常处理程序。

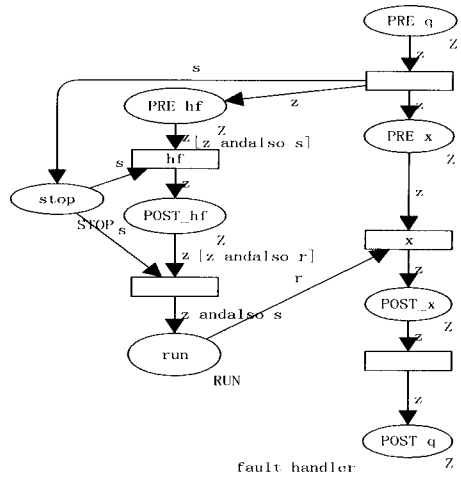


图 3　Faul thandlers 活动的 CPN 模型

3.4　Compensation handlers 的异常处理 CPN 模型

异常处理中涉及到事务处理时，使用补偿的方式来完成。补偿处理器（compensation handler）是用来消除一些在已完成的故障中受影响的工作单元并把数据恢复到执行该工作之前的样子的特定操作。补偿通常是附加在故障处理程序中，先前成功的一些嵌套活动可能需要补偿。一个补偿处理程序通常与一个 SCOPE 相关联（BPEL scope），在 SCOPE 的范围内触发补偿处理。BPEL 允许分别在故障处理器（用 catch 或者 catchall），或者另一个补偿处理器或终止处理器调用补偿活动。

Compensation handlers 可以在以下地方定义：在异常处理的〈catch〉中可以指定补偿的 scopes、补偿域中的〈invoke〉操作对应了 compensation handlers 当 invoke 操作产生异常时、调用 compensation handlers 中定义的补偿操作。图 4 表示 scopes 的补偿活动的着色 Petri 网模型。在 scope（表示为 Q）的一个 Fault handler（表示为 FH）里，调用了一个补偿活动（hc），Compensation handlers 的异常处理 CPN 模型的说明跟 Fault handlers 的类似，可参考第 3.3 节。

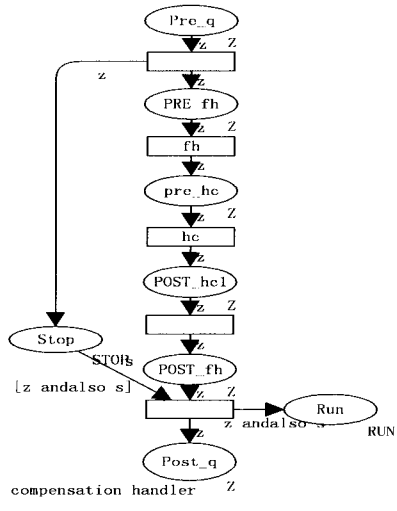


图 4　Compensation handlers 活动的 CPN 模型

3.5　Terminate handlers 的异常处理 CPN 模型

终止处理器（Termination handlers）为 scopes 在一定程度提供了被迫终止的控制语义。Termination handlers 在 scopes 中定义，相当于 java 中的 finally 语句，实现相应的清理工作。BPEL 结构的终止活动在两种情况下执行：因这一个错误而

终止一个错误；在退出活动发生时，终止整个的流程。当一个scope中的活动发生错误时，该scope的主要活动需要终止。根据BPEL规范，当某个并行分支抛出异常时，最内层的异常处理块负责该异常的处理。在执行该异常处理块之前，应首先终止其他所有并行分支的执行，称其为强制终止。当某个scope被强制终止时，其主活动将被强制终止，转而执行其异常处理块。

在BPEL中，终止整个过程通过执行一个过程的退出活动触发。当过程需要终止，所有并发运行的活动必须终止。为映射终止活动，使用STOP库所和RUN库所，这两个库所的具体含义同第3.2节scopes中的。终止处理器的异常处理CPN模型映射如图5所示，在〈scope〉中有一个变迁(活动)X要终止，在执行X变迁前，通过S变迁，传递一个令牌到STOP库所去标示〈scope〉需要终止，异常处理完，当需要执行X时，需要RUN库所传递令牌过来，X才能继续执行。

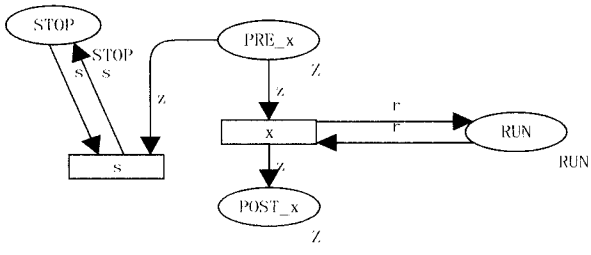


图5 Terminate handlers活动的CPN模型

4 BPEL异常处理CPN模型的转换工具的实现

为了实现本文提出的BPEL流程的异常处理方法，需要为BPEL流程的异常处理提供一个通用的流程层异常处理转换工具去处理在BPEL流程中的错误。如果错误发生在一个流程调用一个活动时，这个工具捕获该活动产生的异常，执行用户指定的定义在异常策略库里中跟该活动异常处理策略关联的规则及动作，在原BPEL代码上添加异常处理语言成份，产生新的带异常处理功能的BPEL代码。

4.1 转换工具的体系结构

BPEL流程层异常处理转换工具的组成由图6给出，主要由以下几部分组成：服务引擎、流程引擎、流程异常检测点植入、异常识别服务、异常通知服务、异常日志服务、异常策略管理、异常动作执行管理、异常识别资源集服务、异常处理资源集服务等。其中服务引擎负责服务组件执行、服务发布注册、服务发现、服务调用等功能。所有的异常相关的处理以服务的形式发布，在服务引擎中，查找并调用相关的异常处理服务，包括异常日志服务、异常通知服务等。流程引擎的主要功能就是执行使用BPEL描述的业务流程，包括流程的部署和流程的管理两部分。流程的部署指的是由执行引擎分析流程建模文件，流程执行引擎部署到相应的环境中去。流程管理是实现业务流程的关键，流程的管理是指实现流程的起、停、部署、卸载、监控等操作。流程异常检测点植入是根据异常的处理策略，为BPEL流程植入异常检测点。利用BPEL流程自身的异常检测功能检测异常的发生。异常策略管理实现策略集中管理，并把异常处理策略发布为服务。策略编辑可以提供策略的编辑、编译器及策略的新增、修改和查询功能。异常资源集服务是存储服务层各类异常及其特征，为异常类型识别提供相关的知识库。异常处理资源集服务提供对异常处理资源集的访问、查询、修改等服务。

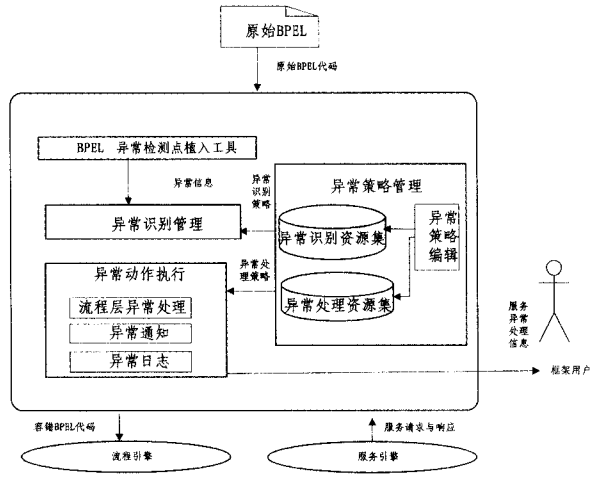


图6 异常处理转换工具的组成

流程异常检测点植入是根据异常的处理策略，为BPEL流程植入异常检测点，利用BPEL流程自身的异常检测功能检测异常的发生。异常识别服务再通过流程异常检测点植入获取消息中的异常信息，根据异常处理资源集的信息进行匹配，判断流程发生的异常的类型，在异常处理资源库中查找能处理该异常类的异常策略，如果有合适的策略来处理发生的异常事件，它就会调用策略执行异常动作执行相应的动作。异常策略库定义了异常条件和它们对应的错误处理动作。异常动作执行服务执行满足条件的策略规则库中的行为，通常每个行为有相应的BPEL代码，异常动作执行服务也会调用异常通知服务把发生的异常信息发送给服务提供者和服务请求者。同时，还要调用异常日志服务把发生的异常信息存入异常日志数据库。

4.2 异常处理转换工具的异常处理逻辑

流程层的异常处理的具体步骤如下：

(1) 它首先通过策略管理库将所需策略及规则设计好(策略管理库有所有用于BPEL流程各种异常处理的策略和实际流程逻辑执行(转换为BPEL代码))，将策略服务化后发布部署到转换工具的集成平台上，并将策略以服务的方式来编排和调用。

(2) 流程异常检测植入服务实时对bpel流程中Web服务调用的活动的执行进行检测，如果检测到异常事件，它就会把异常事件发送给异常识别服务。利用第3.1节第二步介绍的相关原则，在原BPEL代码上添加pre-condition服务的调用和post-condition服务的调用这些跟异常处理相关的语句，并把通过异常识别服务获得的异常类型放进pre-condition服务调用的参数中。

(3) pre-condition服务的调用和post-condition服务的调用会调用对应的异常策略服务，pre-condition服务根据策略服务传过来的异常处理类型，匹配对应的策略并调出策略里相应的规则集；利用第3.1节第三步介绍的相关原则，调用一个策略执行模块将这些规则通过BPEL代码生成服务来完成BPEL语法校验并转换为相应的BPEL代码，嵌入到原BPEL代码中。

(4)最后经过验证的BPEL文件，可以将其部署到BPEL执行引擎中去执行，并通过对流程进行监控，来发现系统运行时刻可能存在的问题。

转换工具中的流程引擎基于开源的流程引擎 Apache ODE，异常策略集使用 WS-Policy 规范，利用 Apache neethi 实现 WS-Policy 规范，通过在 Apache neethi 添加对 AOP 技术的支持，将策略集中的规则转换成 Jboss 公司的规则引擎 drools 的规则文件存放，在 WS-Policy 策略文件里利用 WS-PolicyAttachment 实现对该规则文件的引用。利用 drools 的相关工具集进行异常处理策略和规则的编辑和管理。异常动作执行通过 Drools 提供的 RuleFlow(规则流)实现对规则的编排，通过 Drools 提供的 Drools Fusion 实现对动作的执行。异常日志服务使用 Apache 的 Log4J 框架来记录系统中的所有错误、异常、日志、消息等信息。异常通知服务可基于 WS-Notiflcation 规范去实现。转换工具中的管理监控环境将基于 Flex、Struts、Spring 等 Web 开发框架和技术进行开发，为部署人员提供一个在线的 Web 应用程序。

5 案例研究与分析

5.1 案例说明及其异常处理 CPN 模型

图 7 是一个小型零售供应商供货服务的 BPEL 代码图。整个 BPEL 流程可以看作是一个特殊的 Scope，拥有异常处理程序和流程数据结构。为了节省篇幅，省略了定义声明部分。Scope 里面有一个〈sequnce〉结构活动，包含两个〈invoke〉活动，分别是确认可供货的货物服务调用(InvSupplier AnsweronShop)和中止供货服务调用(Inv-SuspendonShop)。

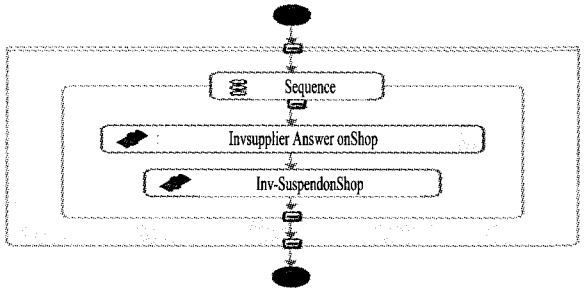


图 7 供应商供货流程的 BPEL 图

图 8 是该供应商 BPEL 流程对应的异常处理的 CPN 模型。变迁 q，su，ex 分别表示图 7 中 BPEL 流程的 Scope 和两个 invoke 活动。假设在执行 Inv-SuspendonShop 的〈invoke〉活动时，产生了一个异常，因此在执行该 invoke 活动时，发送令牌到 STOP 库所，终止正常的流程执行，并从下面的正常活动流程转到上面的异常活动流程；STOP 库所获得令牌后，相继执行 Inv-SuspendonShop 的〈invoke〉活动，执行完后把令牌传给 RUN 库所，恢复了下面正常活动流程。

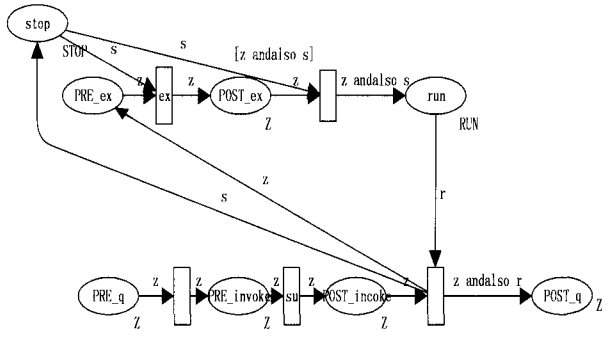


图 8 供应商供货流程的异常处理 CPN 模型

5.2 案例实现思想

为了实现本文提出的 BPEL 异常处理方法，需要在第 4 节提出的通用的异常处理转换工具中处理 BPEL 流程中的错误。该转换工具在用户提供的原 BPEL 代码基础上，在 BPEL 代码的每个会产生异常的活动前加了〈invoke〉活动，调用前置策略服务；在该活动后加了〈invoke〉活动，调用后置策略服务。其中前置策略服务判断该活动会不会产生异常，如果产生异常，还要判断 Scope 里面 stop 相关集的值是否为真，如果为真，则根据相应策略调用相应的异常处理动作，否则会把 stop 相关集设为真，再去调用相应的异常处理动作。后置策略服务判断该活动的异常处理是否正常结束。当正在进行的异常处理活动正常结束时，run 相关集的值为真，stop 相关集的值为假，同时继续执行〈scope〉中其它的活动。

在图 9 的 BPEL 流程中，原始的 BPEL 代码是 1，2，3，5，18，23，24，25，26 行。异常处理转换工具首先在调用第 5 和 18 行的〈invoke〉活动的前后分别添加了两个服务：pre-condition，post-condition，它们是异常处理逻辑的接口。pre-condition 表示检查活动是否有异常处理，并传递相关的参数给后面的异常处理，pre-condition 中的输入参数表示异常的类型，在〈scope〉里定义了两个相关集“stop”和“run”，映射图 8 的 STOP 和 RUN 库所。异常处理器根据异常类型进行相应的处理，如果发现有异常，就把相关集“stop”的值赋为"yes"。post-condition 表示结束一个异常处理，post-condition 中的输出参数表示异常处理的结果，如果异常处理完成，就把相关集“run”的值赋为“yes”，代表异常处理正常结束。

```
1 〈process xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/"〉〈!—原 BPEL 代码-〉
2  〈scope variableAccessSerializable="no"〉〈!—原 BPEL 代码-〉
3   〈sequence〉〈!—原 BPEL 代码-〉
4    〈invoke partnerLink="exceptionpolicy" portType="exception-identify" operation="pre-condition" inputvariable="exceptiontype1"〉
5    〈invoke inputVariable="RealSupplierAnswerMSG" name="InvSupplier AnsweronShop" operation="receiveRealSupplierAnswer" partnerLink="RealSupplier—RealSupplier2Shop" portType="ns2:RealSuppliercallbackPT"〉〈!—原 BPEL 代码-〉
6    〈invoke partnerLink="exceptionpolicy" portType="exception-identify" operation="post-condition" outputvariable="exceptionresult1"〉
7     〈correlationSets〉
8      〈correlationSetname="stop" properties="ns1:RealSupplierCorrelationSet"/〉
9     〈/correlationSets〉
10     〈correlationSets〉
11      〈correlationSetname=e"un" properties="ns1:RealSupplierCorrelationSet"/〉
12    〈/correlationSets〉
13   〈invoke partnerLink="exceptionpolicy" portType="exceptionidentify" operation="pre-condition" inputvariable="exceptiontype2"〉
14     〈correlations〉
15      〈correlationinitiate="yes" set="stop"/〉
16     〈/correlations〉
```

```
17    〈/invoke〉
18    〈invoke inputVariable="RealSupplier" name="Inv-SuspendonShop" operation="ExternalProblemsManagement-RealSupplier" partnerLink="RealSupplier-RealSupplier2Shop" portType="ns2:RealSuppliercallbackPT"/〉〈! —原 BPEL 代码-〉
19    〈invoke partnerLink="exceptionpolicy" portType="exceptionidentify" operation="post-condition" outputvariable="exceptionresult2"〉
20      〈correlations〉
21        〈correlation initiate="yes" set="run"/〉
22      〈/correlations〉
23    〈/invoke〉〈! —原 BPEL 代码-〉
24  〈sequence〉〈! —原 BPEL 代码-〉
25  〈/scope〉〈! —原 BPEL 代码-〉
26 〈/process〉〈! —原 BPEL 代码-〉
```

图 9　供应商供货服务流程的 BPEL 代码

最后，转换工具执行后生成的带异常处理的新的 BPEL 代码如图 10 所示。第 4—10 行代码是该转换工具执行图 9 的第 13 行的 pre-condition 和第 19 行的 post-condition 策略服务后，在原 BPEL 代码基础上生成的额外的代码，其中 4—6 行代码是 pre-condition 服务在策略执行后生成的，而 8—10 行代码是 post-condition 服务对应生成的，由于篇幅限制，省略其它生成代码。

```
1 〈process xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
2  〈scope variableAccessSerializable="no"〉
3   …
4    〈scope exitOnStandardFault="yes"〉〈! —生成的异常处理 BPEL 代码-〉
5     〈faultHandlers〉〈! —生成的异常处理 BPEL 代码-〉
6      〈catchAll〉〈! —生成的异常处理 BPEL 代码-〉
7        〈invoke inputVariable="RealSupplier" name="Inv-SuspendonShop" operation="ExternalProblemsManagement-RealSupplier" partnerLink="RealSupplier-RealSupplier2Shop" portType="ns2:RealSuppliercallbackPT"/〉
8      〈/catchAll〉〈! —生成的异常处理 BPEL 代码-〉
9     〈/faultHandlers〉〈! —生成的异常处理 BPEL 代码-〉
10    〈/scope〉〈! —生成的异常处理 BPEL 代码-〉
11     …
12   〈scope /〉
13 〈/process〉
```

图 10　转换工具生成的带异常处理的 BPEL 代码的片断

结束语　本文提出了一种 BPEL 的异常处理逻辑开发方法。该方法首先利用着色 Petri 网形式化地描述 BPEL 的异常处理机制，具体地建模了故障处理器、事件处理器、补偿处理器和终止处理器的异常处理 CPN 模型。接着在提出的 BPEL 的异常处理 CPN 模型的思想上，提出了一个异常处理转换工具，它将 BPEL 的异常处理 CPN 模型转换成对应的被工具支持的具有异常处理的 BPEL 代码，最后工具对这个带有异常处理的 BPEL 代码进一步解析，将其转换成标准的、完善的异常处理的 BPEL 代码。最后通过供应商供货流程案例展示了本文提出的 BPEL 异常处理建模方法的应用过程，结果表明该方法有利于设计人员在面向服务软件建模过程中显示地分离异常流程和正常流程，增强了 BPEL 异常处理能力，为设计 BPEL 流程提供了一套有效的异常处理建模方法。下一步就该模型方法的一致性、可终止性、完备性等性质提出证明或验证方法，完善基于该方法的异常处理工具。

参 考 文 献

[1] Business Process Execution Language for Web Services version 2.0 [EB/OL]. http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.pdf

[2] Jensen K. Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use[M]. Analysis Methods, Monographs in Theoretical Computer Science, Springer, 1994, 2: 5-10

[3] Cai C, Qiu Z, Yang H, et al. Global-to-Local Approach to Rigorously Developing Distributed System with Exception Handling [J]. Journal Of Computer Science and Technology, 2009, 24(2): 238-249

[4] Carbone M. Session-based Choreography with Exceptions[J]. Electronic Notes in Theoretical Computer Science, 2009, 241(3): 35-55

[5] Yang Y, Tan Q, et al. Transformation BPEL to CP-Nets for Verifying Web Services Composition[C]// Proceeding on the International Conference on Next Generation Web Services practices(NWeSP'05). 2005: 137-142

[6] Verbeek H M W, van der Aalst W M P. Analyzing BPEL processes using Petri nets[C]// Marinescu D C, ed. Proceedings of 2nd International Workshop on Applications of Petri Nets to Coordination, Workflow and Business Process Management. 2005: 59-78

[7] Ouyang Chun, van der Aalst W M P, Breutel S, et al. Formal semantics and analysis of control flow in WS-BPEL[R]. Report BPM-05-15. BPM Center, 2005: 1-34

[8] Hinz S, et al. Transforming BPEL to Petri Nets [C]// Proceedings of 3rd International Conference on BPM. 2005: 220-235

[9] Stahl C. A Petri net semantics for BPEL [R]. Techn. Report 188. Humboldt-Universitat zu Berlin, 2005: 1-84

[10] Hamadi R, Benatallah B, Medjahed B. Self-adapting recovery nets for policy-driven exception handling in business processes [J]. Distributed and Parallel Databases, 2008, 23(1): 1-44

[11] 范贵生，虞慧群，陈丽琼，等. 基于 Petri 网的服务组合故障诊断与处理[J]. 软件学报，2010，21(2)：231-247

[12] 门鹏，段振华. 基于着色 Petfi 网的 BPEL 建模与验证[J]. 西北大学学报，2007，37(6)：986-990

[13] 孙健，张鹏. 基于 Petri 网的 Web 服务流语言(WSFL)建模与分析[J]. 小型微型计算机系统，2004，25(7)：1382-1386