流程违例处理的行为及分析

薛 岗[1] 叶小虎[2] 张 楠[1] 姚绍文[1]

(云南大学软件学院 昆明 650091)[1] (云南大学信息学院 昆明 650091)[2]

摘 要 业务流程运行期间,外部或运行环境中的某些属性与流程设计时所设定的条件不一致时,将导致流程运行时的违例,违例处理方法涉及流程运行时违例的发现和处理。首先介绍了流程的违例及其分类;基于流程违例处理模式,使用CCS偶图对流程工作项级、案例级违例处理以及相关补偿处理进行分析和表述;在违例处理策略讨论的基础上,总结了违例处理模式的基本形式,并通过CCS偶图的动态特征分析了违例处理的动态行为。

关键词 业务流程违例处理,流程违例,流程违例处理分析

Exception Handling Analysis for Business Process

XUE Gang[1] YE Xiao-hu[2] ZHANG Nan[1] YAO Shao-wen[1]

(School of Software Engineering, Yunnan University, Kunming 650091, China)[1]

(School of Information Science and Engineering, Yunnan University, Kunming 650091, China)[2]

Abstract Exceptions are aroused when built-in attributes for executing a well-designed business process are not supported. Process exception handling technology involves detecting and processing runtime exceptions in process executing environment. This paper first introduced features and classification of business process exception. Based on Exception Handling Patterns, bigraphs for CCS were applied to investigating process exception handling, and some results regarding modeling and analysis were discussed. At last, exception handling strategies were applied to analyzing exception handling patterns, and basic model structure and dynamic behavior analysis for process exception handling were proposed in the paper.

Keywords Business process exception handling, Business process exception, Process exception handling analysis

1 引言

业务流程包含大量的活动,它在组织内部和技术环境中运行以实现业务目的。流程的运行环境涉及到了诸如硬件、软件、技术、管理等很多因素。运行时,当外部或运行环境中的某些属性与流程设计时所设定的条件不一致时,将导致流程运行时的违例。违例处理方法涉及流程运行时违例的发现和处理。违例处理模式[1]分别分析了工作项级、案例级(流程实例)的违例处理方法和恢复行为,以文字描述的形式进行总结形成了21种基本处理模式。这些模式可处理工作项失败、过期、资源不可用、外部触发和违反限制等违例。违例处理模式独立于具体的描述方法和技术,可作为衡量流程管理系统违例处理能力的技术标准[1]。

Robin Milner提出的偶图[2]是行为演算[3]的发展,该方法是基于具有位置和链接的图形化计算模型,可以与反应规则合在一起形成偶图反应系统。它的提出是为基于拓扑结构的分布式计算奠定理论基础。该方法提出的目标可对已设计和运行的系统进行建模,而且可通过该理论帮助构建可自动适应未来要求的系统[2]。

本文主要对违例处理技术进行研究,探索违例处理的建模和处理行为分析方法。第2节概述了违例、违例处理、CCS偶图等相关技术;第3节就违例处理的方法进行讨论,并将具体方法进行规范化表述;第4节主要包含违例处理策略和违例处理动态行为的讨论和分析;第5节进行全文总结并说明下一步的工作。

2 相关技术和理论概述

2.1 流程的违例和违例处理

在工作流管理研究中,有研究者把对于违例和违例的产生归纳为:基本错误、应用错误、已知违例和未知违例[4]。随后相关的研究多针对已知违例和未知违例两种情况。对于已知违例,早期一般是采用基于数据库管理技术中事务的方法来进行处理[4]。而对于未知违例的处理,工作流管理技术中则出现了自适应工作流、工作流演化等技术[5]。

违例的发生可以位于活动,也可以发生于流程中。违例的类别可以分为:已知违例和未知违例。对于已知违例,在流程设计阶段可以通过设计明确的捕获过程来截取违例,并进行违例处理。另外,在管理系统设计实现时需提供必要的违例处理功能。对于未知违例,则还需要通过包含系统支持和流程管理等方法来实现违例的处理。本文主要讨论对已知违

到稿日期:2010-12-13 返修日期:2011-03-25 本文受国家自然科学基金项目(60763008)和云南大学(理工)科学研究项目(2009F35Q)资助。

薛 岗(1977—),男,博士,讲师,主要研究方向为工作流管理技术,E-mail:mess@ynu.edu.cn;叶小虎(1988—),男,硕士生,主要研究方向为工作流管理技术;张 楠(1985—),女,硕士生,主要研究方向为计算机网络;姚绍文(1966—),男,博士,教授,主要研究方向为工作流管理技术。

例的处理方法。

2.2 违例处理模式

违例处理模式[1]分别分析了工作项级、案例级(流程实例)的违例处理方法和恢复行为,以文字描述的形式进行总结,形成了21种基本处理模式。这些模式可处理工作项失败、过期、资源不可用、外部触发和违反限制等违例。其中,"工作项失败"主要说明工作项执行过程中出现的错误现象。当工作项执行超过时限要求时,产生"过期违例"。"资源不可用"主要描述当流程无法调用外部资源时产生的违例。"违反限制"是流程属性或运行状态违反设计要求时产生的违例。最后,当涉及到活动的外部触发(即活动的执行依赖于外部活动或事件的触发)时,一般使用"外部触发"来驱动流程的继续运行。

2.3 CCS偶图

一个普通的偶图(例如:G)包含结点,结点之间通过链接进行连接。偶图可分解为两种图形表达形式:位置图(例如:G^P)和链接图(例如:G^L)。位置图表示结点之间的从属关系。链接图中没有从属关系,只表示结点之间的连接关系。一般的偶图需要指明图中元素的接口,接口分为两类:外部接口和内部接口。内部接口可支持与其它偶图进行组合。多个偶图可以根据组合条件进行组合以形成新的偶图;另外,已有的偶图也可以根据需求进行拆分,形成多个新的偶图。偶图的动态特征通过偶图反应系统(Bigraphical Reactive System,BRS)来体现。动态特征的表达需要借助反应规则来说明,反应规则分别指明了系统在变迁之前和之后的具体情况。

偶图的数学基础是范畴(Category),范畴包括两类实体:对象和箭头。这两类实体分别构成对象集和对象集之间的函数(映射)[2]。在偶图中,对象为接口,函数为箭头。一般偶图可以使用二元组来表示,对于偶图 G:$G=\langle G^P,G^L\rangle$,偶图的接口表示为$\langle n,X\rangle$,n为结点总数,X为链接名称集。偶图可为:$B:\langle n,x\rangle\rightarrow\langle m,y\rangle$,其中,$\langle n,X\rangle$是内部接口,$\langle m,Y\rangle$是外部接口。如果存在偶图$F$和偶图$H$,当满足偶图$H$外部接口与偶图$F$的内部接口对应时,可进行组合操作,即:$G=F\circ h$[2]。

对于位置图,没有结点和链接的偶图称为位置(placing)。对于链接图,没有结点和位置的偶图为链接(linking),使用符号λ表示。链接的基本形式为[2]:替换(y/X)、关闭$(/x:x\rightarrow\varepsilon)$。偶图代数元素之间的运算操作有3类[2]:并行($\|$)、嵌套(.)和合并(|)运算。

针对不同的应用,偶图的使用方法也不同,这样必然会产生多种类型的偶图。分类技术支持类型概念,可指定偶图中结点或链接的类型。因此,分类可以分为位置分类和链接分类。一个位置分类$\Sigma=(\Theta,\mathcal{K},\Phi)$,$\Theta$是类型(sorts)集,$\mathcal{K}$是满足$\Theta$的签名。$\Phi$为偶图类型的构建规则,并保证被指定分类的偶图仍然满足偶图的基本性质[2]。

CCS的偶图系统(CCS偶图)是CCS理论[12]的偶图实现系统。通过分析研究表明,CCS偶图除了保留了CCS理论原有的技术特征之外,还具备了偶图的基础特征[2]。CCS的位置分类Σ_{CCS}具有类型$\{p,a\}$和签名:

$$\{alt:(p,0),send:(a,1),get:(a,1)\}$$

式中,p为过程,a为选择[2]。偶图系统中不能含有类型为p的空闲根,且满足[2]:(1)所有具有类型θ的根$r:\theta$的孩子具有类型θ;(2)所有具有类θ型的结点$v:\theta$的孩子具有不同于类型θ的类型。设空(0)为偶图1,且定义原子$nil\overset{def}{=}alt.1$。CCS偶图的抽象参数反应规则是三元组:(R,R',η),其中$R=alt.(send_x|id)|alt.(get_x|id)$,可拥有4个参数;$R'=x|id|id$拥有两个参数;这两个参数由实例映射决定[2]。文献[2]中详细介绍了CCS向偶图映射的方法和两种理论方法之间的关系讨论。

偶图在适普计算[9]、资源访问[6]、XML[7]和业务流程管理[8,11]等领域中得到了应用。另外,该理论的发展目标之一是提供对现存多种过程建模工具的元理论级描述[2,10]。

3 基本违例处理方法的表述

由于篇幅的限制,本部分内容重点阐述基本违例模式的形式化描述。由于偶图支持图形化和代数两种表述形式,两者之间是直接映射关系,本部分主要展示代数的描述结果。关于违例处理模式的详细描述和介绍可参考文献[1]。

3.1 工作项级违例处理

工作项是已被实例化而未处于运行状态的任务。初始化以后的工作项可被提供给系统中的一个或若干个资源执行。资源可根据自身情况选择执行工作项,并可向系统提交执行志愿。系统可根据志愿来给资源分配任务。工作项级的违例处理主要包含15种方式,且本类处理模式主要针对系统中的资源而提出[1]。

Continue-offer(OCO)模式[1]:本模式要求资源收到相应的违例消息以后,不进行任何违例处理,且已提供给资源的工作项还可以正常执行。资源中的触发执行行为和违例处理是并行关系,并且相互不受影响。因此,资源具有触发工作项执行的功能,也有捕获违例或接收违例通知的功能。形式化描述为:

$$Workitem_{oco}=alt.(get_{token}.d_{oco2}|1)$$

$$Resurce_{oco}=/token\circ(alt.(get_{warning}.d_{oco1}|1)|alt.(send_{token}.d_{oco}|1)|Workitem_{oco})$$

系统使用*token*链接实现工作项的执行触发,使用外部链接*warning*实现违例的发现或违例消息。

Reoffer(ORO)模式[1]:本模式要求资源收到相应的违例消息以后,放弃工作项的执行触发,并将已提供的资源再次提供给其它资源。资源中的触发执行和违例处理是选择关系。因此,资源具有触发工作项执行的功能,也有捕获违例或接收违例通知的功能,并可以取消工作项的执行,且可将工作项重新提供给其它资源。形式化描述为:

$$Workitem_{oro}=alt.(get_{token}.d_{oro1}|1)$$

$$Handler_{oro}=alt.(get_{warning}.alt.(send_{halt}.alt.(send_{pass}.d_{oro2}|1)|1)|send_{token}.d_{oro0})$$

$$Offeror_{oro}=alt.(get_{pass}.alt.(send_{cancel}.alt.(send_{end}.d_{oro4})|1)|1)|1)$$

$$Resource_{other}=alt.(get_{cancel}.d_{oro5}|1)$$

$$/L\overset{def}{=}/token\|/pass\|/halt\|/cancel$$

$$Resource_{oro}=/L\circ(Handler_{oro}|Offer_{oro}|Resource_{other}|Workitem_{oro}|alt.(get_{halt}.d_{oro3}|1))$$

系统使用*token*链接实现工作项的执行触发,使用外部链接*warning*实现违例的发现或违例消息,链接*halt*表示取

消工作项的操作，资源还可通过链接 *cancel* 宣布放弃执行工作项。相关违例处理结束以后，资源通过链接 *end* 向外部系统通报处理结束。该描述中，存在一个子过程负责完成工作项的重新提供过程，该子过程通过链接 *pass* 触发。所描述的情况包含了一个外部资源的信息接收子过程，而描述中未涉及到详细的外部资源行为。

Force-complete-o(OFC)模式[1]：本模式要求资源收到相应的违例消息以后，放弃工作项的执行触发，设置工作项的状态为完成。形式化描述为：

$$Workitem_{ofc} = alt.(get_{token}.d_{ofc1}|1)$$

$$Handler_{ofc} = alt.(get_{warning}.alt.(send_{halt}.alt.(send_{completeness}.alt.(send_{next}.d_{ofc2}|1)|1)|1)|send_{token}.d_{ofc0})$$

$$Keeper_{ofc} = alt.(get_{next}.alt.(send_{end}.do_{fc5}|1)|1)$$

$$Suspender_{ofc} = alt.(get_{halt}.d_{ofc3}|1)$$

$$Setter_{ofc} = alt.(get_{completeness}.d_{ofc4}|1)$$

$$/L \overset{def}{=} /token \parallel /halt \parallel /next \parallel /completeness$$

$$Resource_{ofc} = /L \circ (Handler_{ofc}|Workitem_{ofc}|Keeper_{ofc}|Suspender_{ofc}|Setter_{ofc})$$

资源中的触发执行和违例处理是选择关系，相互排斥。资源具有触发工作项执行的功能，也有捕获违例或接收违例通知的功能，并可以取消工作项的执行，且可提示继续该工作项的后续工作。系统使用 *token* 链接实现工作项的执行触发，使用外部链接 *warning* 实现违例的发现或违例消息，链接 *halt* 表示取消工作项的操作，通过 *completeness* 链接实现工作项状态的设置，资源还可通过链接 *next* 提示继续执行工作项的后续工作。相关违例处理结束以后，资源通过链接 *end* 向外部系统通报处理结束。

Force-fail-o(OFF)模式[1]：本模式要求资源收到相应的违例消息以后，放弃工作项的执行触发，设置工作项的状态为失败。资源中的触发执行和违例处理是选择关系，相互排斥。资源具有触发工作项执行的功能，也有捕获违例或接收违例通知的功能，并可以取消工作项的执行，且可终止该工作项的后续所有工作项的执行。形式化描述为：

$$Workitem_{off} = alt.(get_{token}.d_{off1}|1)$$

$$Handler_{off} = alt.(get_{warning}.alt.(send_{halt}.alt.(send_{failure}.alt.(send_{cancel}.d_{off2}|1)|1)|1)|send_{token}.d_{off0})$$

$$Canceler_{off} = alt.(get_{cancel}.alt.(send_{end}.d_{off5}|1)|1)$$

$$Suspender_{off} = alt.(get_{halt}.d_{off3}|1)$$

$$Setter_{off} = alt.(get_{failure}.d_{off4}|1)$$

$$/L \overset{def}{=} /token \parallel /halt \parallel /cancel \parallel /failure$$

$$Resource_{off} = /L \circ (Handler_{off}|Workitem_{off}|Canceler_{off}|Suspender_{off}|Setter_{off})$$

系统使用 *token* 链接实现工作项的执行触发，使用外部链接 *warning* 实现违例的发现或违例消息，链接 *halt* 表示取消工作项的执行，通过 *failure* 链接实现工作项状态的设置，资源还可通过链接 *cancel* 停止执行工作项的后续工作。相关违例处理结束以后，资源通过链接 *end* 向外部系统通报处理结束。

Continue-allocation(ACA)模式[1]：本模式要求资源收到相应的违例消息以后，不进行任何违例处理，且已分配给资源的工作项还可以正常执行。资源中的触发执行和违例处理是嵌套关系。因此，资源具有触发工作项执行的功能，也有捕获违例或接收违例通知的功能。形式化描述为：

$$Workitem_{aca} = alt.(get_{token}.d_{aca1}|1)$$

$$Resource_{aca} = /token \circ (alt.(get_{warning}.alt.(send_{token}.d_{aca0}|1)|1)Workitem_{aca})$$

系统使用 *token* 链接实现工作项的执行触发，使用外部链接 *warning* 实现违例的发现或违例消息。

Reallocate(ARA)和 Reoffer-a(ARO)模式[1]：系统(或是流程执行引擎)可以把任务分配或提供给资源。对行为的描述而言，虽然可以使用不同的链接来实现，但两种行为在描述结果上并无本质区别。从语义上而言，分配过程是根据资源的意愿而进行的；提供过程是系统行为(无需考虑资源的意愿)。另外，CCS 偶图无法表示链接的移动功能，因此，无法表示将工作项从一个资源发送到另一个资源的情况(即无法表示分配和提供的动态过程)。可以使用多个资源共同持有一个工作项的情况，即工作项已被提供或分配给多个资源。这一方面与违例处理模式的表述并无冲突。由于工作项的执行和资源是一对一的关系，因此建模过程中需要使用一定的机制来确保工作项的执行只能被一个资源触发，且触发之前和执行以后通过广播通知所有持有该工作项的资源，未执行工作项的资源必须放弃执行已持有的工作项。在这样的条件下，当一个持有工作项的资源进行违例处理，且被要求放弃执行已持有的工作项时，可通过广播的形式通知其它资源继续该工作项的执行。本文基于模式的表述主要针对工作项被重新分配或提供给(一个)其它资源的情况。对 ARA 和 ARO 的描述结果与 Reoffer(ORO)模式描述相似。而从模式表述上，ARA 和 ARO 的区别仅限于：ARO 支持将一个工作项分配给多个其它的资源。

Force-fail-a(AFF)模式[1]：该模式的形式化描述类似于 OFF 模式的描述，区别在于前者是把工作项分配给资源，而后者是把工作项提供给资源。

Force-complete-a(AFC)模式[1]：该模式的形式化描述类似于 OFC 模式的描述，区别在于前者是把工作项分配给资源，而后者是把工作项提供给资源。

Continue-execution(SCE)模式[1]：本模式支持资源开始执行工作项以后，接收并处理违例。收到相应的违例消息以后，资源不进行任何违例处理，且工作项继续正常执行。资源中的触发执行行为和违例处理是嵌套关系。因此，资源具有触发工作项执行的功能，也有捕获违例或接收违例通知的功能。形式化描述为：

$$Workitem_{sce} = alt.(get_{token}.d_{sce0}|1)$$

$$Resource_{sce} = /token \circ (alt.(send_{token}.alt.(get_{warning}.d_{sce1}|1)|1)|Workitem_{sce})$$

系统使用 *token* 链接实现工作项的执行触发，使用外部链接 *warning* 实现违例的发现或违例消息。

Restart(SRS)模式[1]：本模式支持资源开始执行工作项以后，接收并处理违例。收到相应的违例消息以后，资源终止任务实例，并触发任务的重新执行。资源中的触发执行和违例处理是嵌套关系。因此，资源具有触发工作项执行的功能，也有捕获违例或接收违例通知的功能。形式化描述为：

$$Workitem_{srs}=alt.(get_{token}.(alt.(get_{halt}.alt.(get_{restart}.d_{srs1}|1)|1)d_{srs2})|1)$$

$$/L\overset{def}{=}/token\,\|\,/restart\,\|\,/halt$$

$$Resource_{rsr}=/L\circ(Workitem_{srs}\,|\,alt.(send_{token}.alt.(get_{warning}.alt.(send_{halt}.alt.(send_{restart}.d_{srs0}|1)|1)|1)|1))$$

系统使用 *token* 链接实现工作项的执行触发，使用外部链接 *warning* 实现违例的发现或违例消息。工作项被触发以后在场所 d_{srs2} 处执行具体工作。活动中存在负责完成实例终止操作的并行子过程，通过 *halt* 链接触发，且可以重新开始实例的执行(通过链接 *restart*)。

Reallocate-s(SRA)模式[1]：本模式支持资源开始执行工作项以后，接收并处理违例。收到相应的违例消息以后，资源终止任务实例，并将已分配的资源再次分配给其它资源。资源中的触发执行和违例处理是嵌套关系。资源具有触发工作项执行的功能，也有捕获违例或接收违例通知的功能。形式化描述为：

$$Workitem_{sra}=alt.(get_{token}.(alt.get_{halt}.d_{sra1}|1)|d_{sra4}|1)$$

$$Resource_{other}=alt.(get_{cancel}.d_{sra3}|1)$$

$$Canceler_{sra}=alt.(get_{pass}.alt.(send_{cancel}.alt.(send_{end}.d_{sra2}|1)|1)|1)$$

$$Handler_{sra}=alt.(send_{token}.alt.(get_{warning}.alt.(send_{halt}.alt.(send_{pass}.d_{sra0}|1)1)1)1)$$

$$/L\overset{def}{=}/token\,\|\,/pass\,\|\,/halt\,\|\,/cancel$$

$$Resource_{sra}=/L\circ(Workitem_{sra}\,|\,Resource_{other}\,|\,Handler_{sra}\,|\,Canceler_{sra})$$

资源使用 *token* 链接实现工作项的执行触发，使用外部链接 *warning* 实现违例的发现或违例消息。工作项被触发以后在场所 d_{sra4} 处执行具体工作。活动中存在负责完成实例终止操作的并行子过程，通过 *halt* 链接触发。资源还可通过链接 *cancel* 宣布放弃执行工作项。相关违例处理结束以后，资源通过链接 *end* 向外部系统通报处理结束。该描述中存在一个子过程负责完成工作项的重新提供过程，该子过程通过链接 *pass* 触发。上述模型所描述的情况包含了一个外部资源的信息接收子过程，而描述中未涉及到详细的外部资源行为。

Reoffer-s(SRO)模式[1]：该模式的形式化描述类似于 Reallocate-s(SRA)模式的描述，区别在于前者是把工作项分配给一个资源，而后者是把工作项分配给多个资源。

Force-fail(SFF)模式[1]：本模式支持资源开始执行工作项以后，接收并处理违例。收到相应的违例消息以后，资源终止任务实例，设置工作项的状态为失败，并将已分配的资源再次分配给其它资源。资源中的触发执行和违例处理是嵌套关系。资源具有触发工作项执行的功能，也有捕获违例或接收违例通知的功能。系统使用 *token* 链接实现工作项的执行触发，使用外部链接 *warning* 实现违例的发现或违例消息。工作项被触发以后在场所 d_{sff4} 处执行具体工作。活动中存在负责完成实例终止操作的并行子过程，通过 *halt* 链接触发。资源通过 *failure* 链接实现工作项状态的设置，还可通过链接 *cancel* 停止执行工作项的后续工作。相关违例处理结束以后，资源通过链接 *end* 向外部系统通报处理结束。形式化描述为：

$$Workitem_{sff}=alt.(get_{token}.(alt.(get_{halt}.d_{sff0}|1)d_{sff4})|1)$$

$$Handler_{sff}=alt.(send_{token}.alt.(get_{warning}.alt.(send_{halt}.alt.(send_{failure}.alt.(send_{cancel}.d_{sff1}|1)|1)|1)|1)|1)$$

$$Canceler_{sff}=alt.(get_{cancel}.alt.(send_{end}.d_{sff3}|1)|1)$$

$$Setter_{sff}=alt.(get_{failure}.d_{sff2}|1)$$

$$/L\overset{def}{=}/token\,\|\,/failure\,\|\,halt\,\|\,cancel$$

$$Resource_{sff}=/L\circ(Handler_{sff}\,|\,Workitem_{sff}\,|\,Canceler_{sff}\,|\,Setter_{sff})$$

Force-complete(SFC)模式[1]：本模式类似于 SFF 模式表示。形式化描述为：

$$Workitem_{sfc}=alt.(get_{token}.(alt.(get_{halt}.d_{sfc0}|1)d_{sfc4})|1)$$

$$Handler_{sfc}=alt.(send_{token}.alt.(get_{warning}.alt.(send_{halt}.alt.(send_{completeness}.alt.(send_{cancel}.d_{sfc1}|1)|1)|1)|1)|1)$$

$$Keeper_{sfc}=alt.(get_{next}.alt.(send_{end}.d_{sfc3}|1)|1)$$

$$Setter_{sfc}=alt.(get_{completeness}.d_{sfc2}|1)$$

$$/L\overset{def}{=}/token\,\|\,/completeness\,\|\,/halt\,\|\,/next$$

$$Resource_{sfc}=/L\circ(Handler_{sfc}\,|\,Workitem_{sfc}\,|\,Keeper_{sfc}\,|\,Setter_{sfc})$$

本模式支持资源开始执行工作项以后，接收并处理违例。收到相应的违例消息以后，资源终止任务实例，设置工作项的状态为完成。资源中的触发执行和违例处理是嵌套关系。资源具有触发工作项执行的功能，也有捕获违例或接收违例通知的功能。该模式的描述使用 *token* 链接实现工作项的执行触发，使用外部链接 *warning* 实现违例的发现或违例消息。工作项被触发以后在场所处执行具体工作。活动中存在负责完成实例终止操作的并行子过程，通过 *halt* 链接触发。通过 *completeness* 链接实现工作项状态的设置，资源还可通过链接 *next* 提示继续执行工作项的后续工作。相关违例处理结束以后，资源通过链接 *end* 向外部系统通报处理结束。

3.2 案例级违例处理与恢复行为

违例的处理还涉及到案例(流程实例)级违例处理，一般可以通过两种方式来实现：通过提供额外的活动或子过程来完成违例处理的后继工作。以运行系统的观点，流程实例级别的违例处理需要实现[1]：继续工作流案例(continue workflow case，CWC)，删除当前工作流案例(remove workflow case，RCC)和删除所有工作流案例(remove all case，RAC)。

这 3 类模式所表述的行为主要针对系统执行引擎而言，而具体的处理细节是根据实际情况而定。本文将这 3 类模式抽象表示为 $alt.(get_x.d|1)$，其中链接可用不同的命令来替代，例如 *cwc*，*rcc* 或 *rac*。

系统完成违例处理，还可以采取一些额外动作帮助恢复违例的执行，主要有[1]：无作为(no action，NIL)，回滚(rollback，RBK)和补偿(compensate，COM)。

这 3 类模式所表述的行为主要针对系统执行引擎而言，除了无作为模式以外，具体的处理细节是根据实际情况而定。因此，本文将后两类模式表示为 $alt.(get_y.d|1)$，其中链接可用不同的命令来替代，例如：*rbk* 或 *com*。

4 违例处理的策略和动态行为

4.1 违例处理策略

假定在流程中工作项运行过程的某个时刻发生违例。违例之间相互区别，且可被系统侦测并识别时，相应的处理依赖于违例的类型。违例的处理策略主要需要考虑：工作项的处理方法；与工作项相关的案例处理方法；采取什么样的恢复行为以减少违例带来的损失。

由此，文献[1]将工作项级违例处理、案例级违例处理和恢复行为进行综合考虑，最终可形成135种可能的违例处理模式，经过分析可应用于实践的违例处理模式共有36种。其中，模式名称表示了该模式的处理事项，例如：SFF-CWC-COM处理模式规定当一个工作项开始执行以后，状态错误导致违例产生并被捕获时，该工作项被终止，状态设置为失败；系统调用补偿过程；案例中的其它工作项被继续触发执行。

基于这36种处理模式及本文第3节的描述，可以归纳总结出：

· 资源对工作项的处理：

a)初始条件为：系统分配或提供工作项给资源，工作项具有“等待执行”和“已经开始执行”两种状态；

b)对工作项的处理有：不作为(对工作项在资源的执行不做任何干预)和“取消工作项”；

c)对工作项状态设置为：不设置、设置为“失败”和设置为“成功”；

d)对工作项的后续处理有：不作为、“流程中的所有后续工作项不能被触发”、“所在流程中的所有后续工作项被触发”和(通过提供或分配)放弃执行工作项；

e)处理结果可以被汇报或不被汇报。

· 系统对流程实例的处理：继续执行、删除案例(或所有案例)。

· 系统补偿行为：不作为、回滚和补偿。

假设系统所发生的违例可以被侦测和捕获的情况下，违例处理顺序为：

(1)流程执行过程中发生违例情况，发生的位置可以是活动或流程实例；

(2)系统捕获违例并进行分析；

(3)系统对活动实例进行违例处理，即实施工作项级违例处理；

(4)实施流程补偿行为；

(5)对流程实例进行违例处理，即实施案例级违例处理。

设计实现时，系统需要为违例的发现和处理建立独立处理功能。本文使用一个过程模型来获取违例，并按上述策略来进行违例处理。在不考虑违例分析时，一个简单的违例处理模型为：

$$alt.(get_{exception}.alt.(send_{warning}.(alt.(send_{ch}.d_{ch}\,|1)\,|alt.(send_{ra}.d_{ra}\,|1))\,|1)\,|1)$$

模型通过 *exception* 获取违例以后通过 *warning* 触发工作项级违例处理。最后，使用链接 *ch* 触发案例级违例处理和使用 *ra* 触发恢复行为。恢复行为和案例级违例处理被设计成并行触发。这样的机制支持将恢复行为设计成为流程外部行为。如果需要考虑执行的顺序，则可使用链接来进行处理状态的通知，如下所示($Hand_{lercase}$ 和 $Action_{recovery}$ 为案例处理及恢复行为，工作项级违例处理完毕发送 *end* 表示处理结束)：

$$(alt.(send_{warning}\,|1)\,|alt.(get_{end}.(Handler_{case}\,|Action_{recovery})\,|1)$$

另外，上述模型中没有考虑违例分析的过程。分析过程属于违例处理模型的内部行为，外部不可见。若需要对内部行为进行建模，可以在系统内部使用：

$$/x\circ(alt.(send_x\,|1)\,|alt.(get_x.d\,|1))$$

在3.1节中所描述的处理模型中，工作项一般都不包含违例的发生表示。模型中的工作项一般使用场所来抽象工作项触发以后的后续工作：$alt.(get_{token}.d\,|1)$。如果需要对工作项的后续工作再进行细化，同时支持工作项执行过程中违例的发生，可使用：$alt.(send_{exception}.d_0\,|d_1)$来说明违例情况的发生。其中 d_1 是活动正常的工作行为，d_0 为活动违例发生以后的后继行为。根据偶图的组合定义，可表示具有违例掷出功能的工作项为：

$$alt.(get_{token}.d\,|1)\circ alt.(send_{exception}.d_0\,|d_1)=alt.(get_{token}.alt.(send_{exception}.d_0\,|d_1\,|)1)$$

外部违例触发事件直接使用 $alt.(send_{exception}.d_0\,|d_1)$ 表示。其中，d_1 和 d_0 为参数。

4.2 违例处理的动态行为

本部分使用36种违例处理模式中的OCO-CWC-NIL模式作为主要的讨论对象。模式可表述为：

$$Handler_{oco\text{-}cwc\text{-}nil}=alt.(get_{exception}.alt.(send_{warning}.alt.(send_{cwc}.d_0\,|1)\,|1)\,|1)$$

$$Handler_{cwc}=alt.(get_{cwc}.d_1\,|1)$$

$$/L\overset{def}{=}/warning\,\|\,/cwc$$

$$Pattern_{oco\text{-}cwc\text{-}nil}=/L\circ(Handler_{oco\text{-}cwc\text{-}nil}\,|Resource_{oco}\,|Handler_{cwc})$$

该模式规定工作项已被提供给一个或多个资源，违例发生以后，该工作项状态没有发生改变并可以继续执行；系统继续工作流案例；违例处理完毕以后，系统不采用任何补偿措施。根据OCO模式的描述，$Resource_{oco}$ 使用作为支持工作项级违例处理的资源；建立违例级违例处理功能 $Handler_{cwc}$，并使用违例处理过程 $Handler_{oco\text{-}rcc\text{-}nil}$。

为了分析上述模型，假设系统中存在违例被掷出的过程，则可分析模型为：

$$SYS_{oco\text{-}cwc\text{-}nil}=/exception\circ(alt.(send_{exception}\,|1)\,|Pattern_{oco\text{-}cwc\text{-}nil})$$

模型可在反应规则的指导下进行动态变化，$SYS_{oco\text{-}cwc\text{-}nil}$ 的执行存在4种变迁序列，具体过程使用代数形式表示为(其中，*SYS* 为 $SYS_{oco\text{-}cwc\text{-}nil}$ 的缩写)：

$$SYS\rightarrow\begin{cases}SYS_1^0\rightarrow SYS_2^0\rightarrow SYS_3^0\rightarrow SYS_4^0\\ SYS_1^1\rightarrow\begin{cases}SYS_2^1\rightarrow SYS_3^1\rightarrow SYS_4^1\\ SYS_2^2\rightarrow\begin{cases}SYS_3^2\rightarrow SYS_4^2\\ SYS_3^3\rightarrow SYS_4^3\end{cases}\end{cases}\end{cases}$$

上述变迁过程中，状态 SYS_4^0、SYS_4^1、SYS_4^2 和 SYS_4^3 具有相同的表示形式，即系统不能再进行变迁时的状态。上述变迁序列中存在一条变迁序列为：

$$SYS\rightarrow SYS_1^1\rightarrow SYS_2^2\rightarrow SYS_3^3\rightarrow SYS_4^4$$

详细的模型表述为：

$$SYS=/exception\circ(alt.(send_{exception}\,|1)\,|/L\circ(alt.(get_{exception}.alt.(send_{warning}.alt.(send_{cwc}.d_0\,|1)\,|1)\,|1)\,|Reseour_{oco}\,|Handler_{cwc}))$$

$SYS_1^1 = /exception \circ (exception | /L \circ (alt.(send_{warning}.alt.(send_{cwc}.d_0 | 1) | 1) | Resource_{\alpha o} | Handler_{cwc}))$

$SYS_2^2 = /exception \circ (exception | /L \circ (warning | alt.(send_{cwc}.d_0 | 1) | /token \circ (d_{\alpha o1} | alt.(send_{token} \circ d_{\alpha o0} | 1) | Workitem_{\alpha o}) | Handler_{cwc}))$

$SYS_3^3 = /exception \circ (exception | L \circ (warning | cwc | d_0 | /token \circ (d_{\alpha o1} | alt.(send_{token}.d_{\alpha o0} | 1) | Workitem_{\alpha o}) | d_1))$

$SYS_4^3 = /exception \circ (exception | /L \circ (warning | cwc | d_0 | /token \circ (token | d_{\alpha o1} | d_{\alpha o0} | d_{\alpha o2}) | d_1))$

该序列表示系统掷出违例(通过 *exception* 链接),系统变为 SYS_1^1;之后通过 *warning* 链接通知资源进行工作项级违例处理,系统变为 SYS_2^2(由于 OCO 模式不对工作项产生实际影响,故剩余工作为:$d_{\alpha o1}$,详细动作可根据实际情况细化);系统再通过 *cwc* 链接触发案例级别违例处理,系统变为 SYS_3^3(具体工作为:d_0,详细动作可根据实际情况细化);最后,系统继续执行工作项(通过 *token* 触发工作项,具体工作为:$d_{\alpha o2}$)。根据 OCO 模式的描述,本文中的 $Resource_{\alpha o}$ 将违例的通知与工作项的触发设计为并行方式,所以系统中资源获得违例通知与工作项的触发没有先后顺序。由此其他 3 条变迁序列分别为:

- 系统 $SYS_{\alpha o\text{-}cwc\text{-}nil}$ 先触发工作项(系统变迁至 SYS_1^2 状态),之后再进行违例的掷出、捕获,最后执行后续违例处理(具体包含工作项级违例处理和案例级违例处理);
- 系统也可能在捕获违例之后,直接触发工作项(系统变迁至 SYS_2^1 状态),之后再进行后续违例处理(具体包含工作项级违例处理和案例级违例处理);
- 系统捕获违例,完成工作项级别违例处理,在执行案例级违例处理以前,进行工作项的触发(系统变迁至 SYS_3^2 状态),之后再进行案例级违例处理。

最后,由于 OCO-CWC-NIL 模式不要求进行流程补偿处理,因此上述模型的动态分析未涉及违例处理中的补偿处理。

结束语 本文首先介绍了流程的违例及其分类;针对已知违例的情况,论文使用 CCS 偶图对流程工作项级、案例级违例处理,以及相关补偿处理进行分析和表述;基于违例处理策略的讨论,本文总结了违例处理模式的基本形式,并通过 CCS 偶图的动态特征分析了违例处理的动态行为。

未来的研究工作可以从以下几个方面展开:首先,针对流程中的未知违例进行研究,总结未知违例的技术特点;其次,对未知违例的处理提出新的处理策略和方法;最后,将违例处理的策略和方法应用于灵活的流程管理应用系统中。

参 考 文 献

[1] Russell N, van der Aalst W M P, ter Hofstede A H M. Workflow Exception Patterns[C]// Proceedings of the 18th International Conference on Advanced Information Systems Engineering(CAiSE 06). vol. 4001 of LNCS, Berlin: Springer-Verlag, 2006:288-302

[2] Milner R. The space and motion of communicating agents[M]. Cambridge University Press, April 2009

[3] Milner R. Calculi for interaction[J]. Acta Informatica, 1996: 707-737

[4] Eder J, Liebhart W. The workflow activity model (WAMO)[C]// Proceedings of the 3rd international conference on Cooperative Information Systems (CoopIs). Vienna, Austria, May 1995

[5] Rinderle S, Reichert M, Dadam P. Correctness criteria for dynamic changes in workflow systems-a survey[J]. Data and Knowledge Engineering, 2004, 50: 9-34

[6] Grohmann D, Miculan M. Controlling resource access in Directed Bigraphs[C]// Proceedings of the 7th International Workshop on Graph Transformation and Visual Modeling Techniques (GT-VMT 2008), In Electronic Communications of the EASST (Vol. 10). European Association of Software Science and Technology

[7] Conforti G, Macedonio D, Sassone V. Bigraphical Logics for XML[C]// Proceedings of the 13th Italian Symposium on Advanced Datebase Systems (SEBD). 2005: 392-399

[8] Bundgaard M, et al. Formalizing WS-BPEL in Binding Bigraphs [R]. May 2008. http://espen.dk/papers/BPELinBind ing BigraphsWSFM.pdf

[9] Birkedal L, Bundgaard M, Damgaard T C, et al. Bigraphical programming languages for pervasive computing[C]// Proceedings of the International Workshop on Combining Theory and Systems Building in Pervasive Computing. 2006: 653-658

[10] Jensen O H, Milner R. Bigraphs and mobile processes (revised) [R]. UCAM-CL-TR-580. University of Cambridge Computer Laboratory, 2004

[11] Hildebrandt T, Niss H, Olsen M. Formalising business process execution with bigraphs and reactive XML[C]// Proceedings of the 8th International Conference on Coordination Models and Languages. vol. 4038 of LNCS, Springer Verlag, 2006: 113-129

[12] Milner R. A Calculus of Communicating Systems [J]. LNCS, 1980, 94

[13] 巫茜,张栋,包坤. 基于角色的工作流平台访问控制安全模型[J]. 重庆理工大学学报:自然科学版,2011,25(3):78-82

(上接第 126 页)

[2] Fu X, Bultan T, Su J. Analysis of Interacting BPEL Web Service [C]// Proc. of the 13th International Conference on the World Wide. New York, USA, 2004: 621-630

[3] Diaz G, Pardo J J, Cambronero M E, et al. Automatic Translatiion of WS-CDL Choreographies to Timed Automata[C]// Proc. of the International Workshop on Web Services and Formal Methods. LNCS, 3670. Versailles, France, 2005: 230-242

[4] Zhao Xiang-peng, Yang Hong-li, Qiu Zong-yan. Towards The Formal Model and Verification of Web Service Choreography Description Language[C]// Proc. of the 3th International Workshop Web Services and Formal Method (WS-FM 2006). 2006: 273-287

[5] Guermouche N, Godart C. Timed Model Checking Based Approach for Web Services Analysis[C]// Proc. of the 7th Int'l Conf. on Web Services. Los Angeles, USA, 2009: 213-221

[6] 门鹏,段振华. 着色 Petri 网模型检测工具的扩展及其 Web 服务组合中的应用[J]. 计算机研究与发展,2009,46(8):1294-1303

[7] 骆翔,字陈艳,古天龙,等. 基于时态认知逻辑的 Web 服务模型检测[J]. 计算机科学,2009,36(8):153-157

[8] 何亚丽,戎玫,张广泉. 一种基于 UPPAAL 的 Web 服务组合模型检测方法[J]. 计算机科学,2010,37(11):122-125