一种支持组合事务的执行语义分析方法

梅晓勇[1,2],李师贤[1],黄昌勤[3],郑小林[4]
(1.中山大学信息科学与技术学院,广东广州 510006;2.湖南文理学院计算机学院,湖南常德 415000;
3.加利福利亚州立大学欧文分校,Irvine 92697;4.浙江大学计算机科学与技术学院,浙江杭州 310027)

摘 要: 为了保证组合事务应用需求,不可避免引入失败恢复机制,以提供可靠的执行语义.本文提出一种基于向前、向后和替代恢复的综合事务恢复机制的执行语义分析技术,借助 Petri 网的动态执行推演技术和数据流分析技术,讨论聚合模式执行语义,最终实现组合事务失败恢复行为的无缝添加/删除.通过对旅行预订组合事务实例分析,表明该语义分析方法是可行的.

关键词: 组合事务;失败恢复机制;执行语义分析;扩展 Petri 网
中图分类号: TP302 **文献标识码:** A **文章编号:** 0372-2112 (2012) 07-1386-11
电子学报 URL: http://www.ejournal.org.cn **DOI:** 10.3969/j.issn.0372-2112.2012.07.017

An Execution Semantic Analysis Method for Composition Transaction

MEI Xiao-yong[1,2], LI Shi-xian[1], HUANG Chang-qin[3], ZHENG Xiao-lin[4]
(1. *School of Information Science and Technology, Sun Yat-sen University, Guangzhou, Guangdong* 510275, *China*;
2. *School of Computer Science and Technology, Hunan University of Arts and Science, Changde, Hunan* 415000, *China*;
3. *Department of Electrical Engineering and Computer Science, University of California, Irvine, CA*, 92697, *USA*;
4. *College of Computer Science and Technology, Zhejiang University, Hangzhou, Zhejiang* 310027, *China*)

Abstract: To assure application requirements of composition transaction, it is inevitable fact for appropriate failure recovery mechanisms that can provide reliable execution semantics. An execution semantic analysis method based on comprehensive transaction recovery mechanism including forward recovery, backward recovery and alternative recovery is proposed, by means of dynamic execution reasoning and data flow analysis of extended Petri nets, which constructs seamlessly Add/Remove recovery behavior for composition transaction. Finally, an application case with recovery capacity is implemented and shows the proposed semantic analysis method is feasible.

Key words: composition transaction; failure recovery mechanism; execution semantic analysis; extended Petri nets

1 引言

Web 服务环境中,存在多种不可预知因素导致长运行事务 LRTs(Long Running Transactions)执行失败的几率更高,如何确保可信组合服务执行,构建可信组合事务处理机制成为影响 LRTs 可靠执行的关键问题.现有的组合事务机制大都不支持多事务恢复性质,缺乏对组合事务松弛 ACID 性质的执行语义分析,难以保证系统执行语义的一致性.

目前关于 LRTs 的执行语义还有待进一步研究,特别是组合务事务执行失败时,需要有效的失败恢复机制处理失败发生,并准确分析失败恢复的执行语义,为 LRTs 的可靠执行和优化奠定基础.目前,关于组合事务的执行语义研究还较少,大多研究组合事务的失败补偿和逆向恢复[1~3],以及形式化建模[4,5]和验证[6,7].由于组合事务执行的动态和随机性,涉及执行日志的迹分析,难以选择性价比高的恢复策略满足失败恢复需求,这势必影响组合事务的执行效果,然而,目前有关组合事务的执行语义研究还较少,Wiesner 等分析了顺序和并行聚合的补偿语义[8,9].Lakhal 提出一种支持组合服务失败执行的语义分析框架 FENECIA[1].组合事务的执行语义分析可以更准确地描述失败恢复,具体选择合适的恢复策略,而不仅仅从框架上给出一个抽象定义,局限于失败补偿,缺乏对恢复通用方法的研究.

本文引入 Petri 网建模组合事务的失败恢复,并讨论其的执行语义.选择 Petri 网建模和分析失败恢复语

收稿日期:2010-11-12;修回日期:2012-02-16
基金项目:国家科技攻关计划(No.2008BAH24B03);国家自然科学基金(No.60673122,No.60940033);中国博士后科学基金(No.20080440121);广东省自然科学基金(No.06017089,No.10JJ6100);广东省科技计划基金(No.2010GK3020)

义,出于以下原因几个点考虑:(1)其具有良好的形式化建模和分析技术;(2)具有良好的图形化表示;(3)Petri 网比有限状态机、过程代数等在面向失败补偿和恢复形式化方面提供了一个更宽基础[11,12].本文通过定义格局变迁来研究分析 LRTs 执行语义,指定执行格局,定义聚合模式的事务执行范型,通过符号方式变换讨论聚合模式的执行语义,得到一系列描述组合事务执行模型.不同聚合模式对同一执行动作可能得到不同的解释,从而避免各聚合模式执行语义的不一致性.

2　组合事务模型

2.1　组合事务的形式化

定义 1　组合事务可定义为一个五元组 $TCS=(I,\lambda,\alpha,\beta,\gamma,m)$,其中:

(1)I 表示为三元组(P,T,F),其中:

(ⅰ)$P=P^s\cup P^{io}\cup P^{qos}\cup P^c$,$P^s$ 为 I 的状态库所集 $P^s\in\{Ready, Activated, Running, Failed, Aborted, Cancelled, Committed, Compensated, Half\_Compensated\}$;$P^{io}$ 表示 I 的输入或输出数据库所集,描述 WS 的功能性指标.P^{qos} 表示 I 的 QoS 库所集,描述 WS 的非功能性指标.P^c 表示控制库所集.下文分别用 $I.p^s$,$I.p^{io}$ 和 $I.p^{qos}$ 获取任务 I 的状态、功能性和非功能性参数;

(ⅱ)$T=T^n\cup T^b\cup\tau$,其中 T^n 表示 I 上正向变迁集$\{Activate(), Run(), Abort(), Cancel(), Commit(), Fail()\}$,$T^b$ 表示 I 上反向变迁集$\{Retry(), Compensate(), Hcompensate()\}$,$\tau\in T$ 不执行任何操作,仅辅助流程.事件 $Activate()$,$Run()$,$Fail()$,$Retry()$,$Abort()$,$Compensate()$,$Commit()$,$Cancel()$简记为 t^{act},t^{run},t^{fal},t^{rty},t^{abt},t^{cmt},t^{cmp},t^{cnl},用 $I.t$ 获取任务 I 的动作;

(ⅲ)$F=(P\times T)\cup(T\times P)$描述 P 到 T 或 T 到 P 的控制流和数据流集合.观察动作/状态对(t_1,p_1)和(t_2,p_2)之间的约束关系:$(t_1,p_1)\prec(t_2,p_2)$、$(t_1,p_1)\lhd\rhd(t_2,p_2)$和$(t_1,p_1)\approx(t_2,p_2)$,其中$(t_1,p_1)\prec(t_2,p_2)$表示动作 t_1 先于 t_2 发生;$(t_1,p_1)\lhd\rhd(t_2,p_2)$表示这动作 t_1 和 t_2 是相互排斥的;$(t_1,p_1)\approx(t_2,p_2)$表示动作 t_1 和 t_2 都发生或都不发生.这些关系分别具有一些有趣的特性,$\prec$是反对称和传递,也就是说,$((t_1,p_1)\prec(t_2,p_2))\Rightarrow((t_2,p_2)\prec(t_1,p_1))$和$(((t_1,p_1)\prec(t_2,p_2))\wedge((t_2,p_2)\prec(t_3,p_3)))\Rightarrow(t_1,p_1)\prec(t_3,p_3)$,而$\lhd\rhd$是反自反、对称和反传递的.除此之外,还可推出性质:

①$((t_1,p_1)\prec(t_2,p_2))\wedge((t_2,p_2)\lhd\rhd(t_3,p_3))\Rightarrow((t_1,p_1)\lhd\rhd(t_3,p_3))$;

②$(((t_1,p_1)\prec(t_2,p_2))\wedge((t_1,p_1)\lhd\rhd(t_3,p_3)))\Rightarrow((t_2,p_2)\lhd\rhd(t_3,p_3))$;

③$(((t_1,p_1)\prec(t_2,p_2))\wedge((t_2,p_2)\approx(t_3,p_3)))\Rightarrow((t_1,p_1)\prec(t_3,p_3))$;

④$(((t_1,p_1)\prec(t_2,p_2))\wedge((t_1,p_1)\approx(t_3,p_3)))\Rightarrow((t_2,p_2)\prec(t_3,p_3))$;

⑤$(((t_1,p_1)\approx(t_2,p_2))\wedge((t_2,p_2)\lhd\rhd(t_3,p_3)))\Rightarrow((t_1,p_1)\lhd\rhd(t_3,p_3))$;

⑥$(((t_1,p_1)\approx(t_2,p_2))\wedge((t_1,p_1)\lhd\rhd(t_3,p_3)))\Rightarrow((t_2,p_2)\lhd\rhd(t_3,p_3))$.

I' 提交是其补偿前提条件,那么$(Commit(), Committed)\prec(Compensate(), Compensated)$;而提交和中止,则$(Commit(), Committed)\lhd\rhd(Abort(), Aborted)$.

(2)λ 描述 I_i 和 I_j 之间的依赖,数据流依赖包括:显式依赖 $I_i\gg^{exp}I_j$ 和隐式依赖 $I_i\gg^{imp}I_j$;行为依赖包括:激活依赖 $I_j\xrightarrow{Act}I_i$ 中止依赖 $I_j\xrightarrow{Abt}I_i$,取消依赖 $I_j\xrightarrow{Cnl}I_i$ 和补偿依赖 $I_j\xrightarrow{Cpt}I_i$.聚合依赖:*And-Split* 聚合依赖$(I_j\parallel I_k)\xrightarrow{Cpt}I_i$,*And-Join* 聚合依赖 $I_k\xrightarrow{Cpt}(I_i\parallel I_j)$,*Or-Split* 聚合依赖$(I_j\otimes I_k)\xrightarrow{Cpt}I_i$,*Or-Join* 聚合依赖 $I_k\xrightarrow{Cpt}(I_i\otimes I_j)$.

(3)$\alpha:I\to Type$ 为聚合映射,其中 $Type\in\{sequence(\oplus), select(\otimes), parallel(\parallel), dicriminator(\Theta), loop(\lfloor)\}$.

(4)$\beta:I\to state$ 为状态映射函数,其中 $state\in\{initial, active, failed, completed, aborted, cancelled\}$,$TCS$ 的状态依据任务的执行进展而定,记为 $CS.state=\bigcup_{i=1}^{n}(I_i.state)$.

(5)$\gamma:I\to TProperty$ 为任务的事务性质映射函数,其中 $TProperty\in\{Vital, Compensable, Retriable, Pivot\}$.组合事务性质 $CS.TProperty=\bigcup_{i=1}^{n}(I_i.TProperty)$.

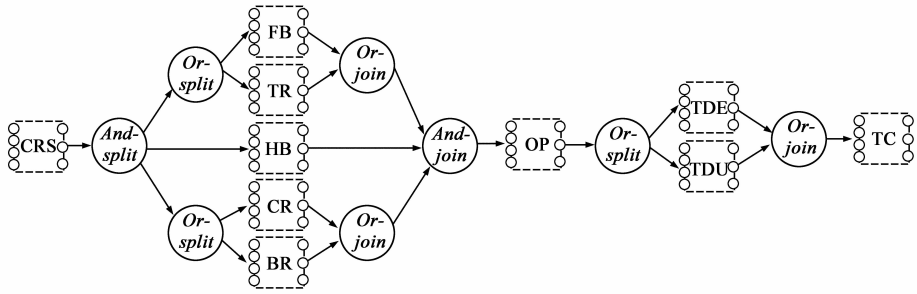

图1　旅行预订组合事务流程

(6) $m: I \rightarrow I'$,当 TCS 中已任务均处于提交状态,而执行失败任务又是不可重试的,于是触发向后恢复,恢复已提交任务,补偿完成后置为已补偿的状态.

如图 1 所示,我们以经典的旅行预订流程 TRP 为例,首先旅行者依据行程规划形成需求规格说明 CRS 提交给旅行社;依据 CRS,分别购买飞机票 FB 或预订火车票 TR;预订宾馆 HB;租赁小汽车 CR 或巴士 BR;预定确认后,旅行者执行在线支付 OP;选择快递公司 TDE (或 TDU)投递给旅行者,将成功预订旅行发送至旅行者确认 TC. 利用聚合算子的嵌套,该组合事务流程为 $CRS \oplus ((FB \otimes TR) \parallel HB \parallel (CR \parallel BR)) \oplus OP \oplus (TDE \otimes TDU) \oplus TC$.

2.2 原子 Web 事务

每个原子 Web 服务都有独特的事务行为,其事务行为与功能语义性质密切相关,分为四类:枢轴(*pivot*) Web服务(记为 WS^p)、可补偿(*Compensable*) Web 服务(记为 WS^c)、可重试(*Retriable*)Web 服务(记为 WS^r)和关键(*Vital*) Web 服务(记为 WS^v),记为 $TBP(WS)$,其中 $TBP(WS) \in \{Vital, Compensable, Retriable, Pivot\}$.

WS^p 是指 Web 任务既不是可重试的,也不是可补偿的,Web 任务一旦成功执行,其产生的影响将不能从语义上消除. 如图 2(*a*)所示.

WS^c 是指对于已成功提交的 Web 任务 I^c,可调用其对应的补偿任务语义上消除所产生的影响. WS^c 的事务行为和 WS^p 不同的是前者具有事务恢复能力,如图 2(*b*)所示.

WS^r 是指 Web 任务 I^r 有限次执行后,确保成功执行. WS^r 的事务行为和 WS^p 不同的是前者具有重试能力,如图 2(*c*)所示.

WS^v 同时具备可重试服务和可补偿服务的能力. 如图 2(*d*)所示,其同时具有 WS^c 和 WS^r 的事务行为.

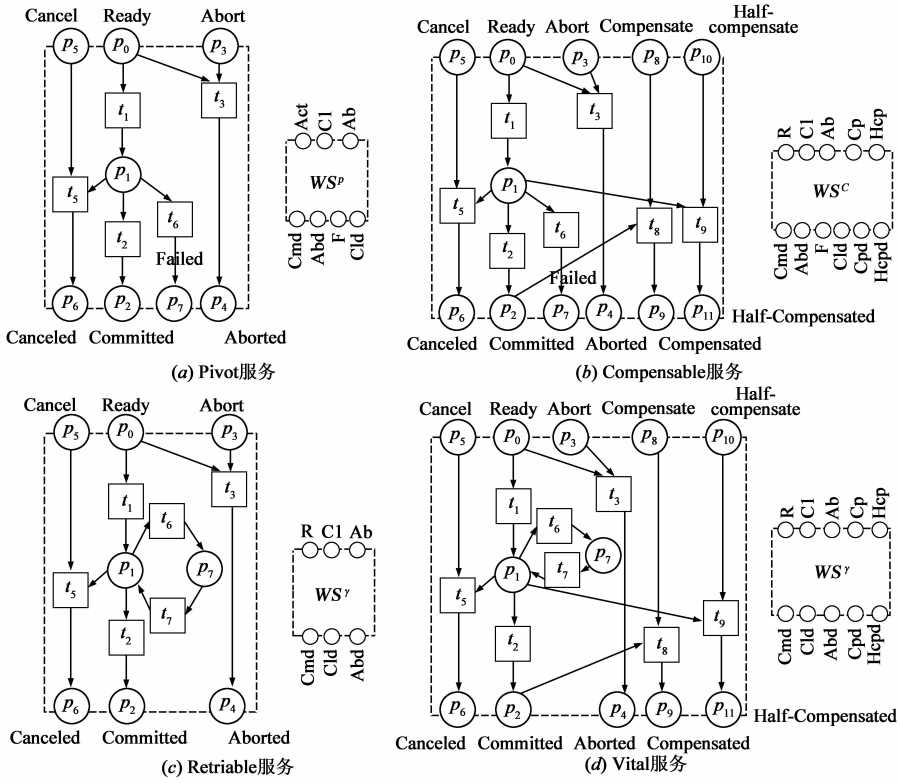

图2 原子Web的状态转移图

2.3 执行日志

组合服务执行时,需为每一任务创建相应的日志,记录每一事务操作,仅当组合服务的根事务提交,日志才会最终提交给执行引擎,而子事务提交时,其对应日志仅记录在其私有缓冲区中. 下面给出组合事务执行日志的形式化描述:

设组合事务 TCS 的执行日志迹 $\sigma = el_1 el_2 \cdots el_n \in Log^*$,其中 el_i 对应 TCS 中各子任务的执行日志记录 $ActID, Desc, QoS, TBP, Behavior, State, \Gamma, In, Out, dom(\sigma) = \{1,2,\cdots,n\}$ 为 σ 的域,满足如下性质:

(1)若 σ_i 是 TCS 的执行日志迹,补偿执行迹 σ_{i_j} 是 σ_i 的一个前缀投影,满足: $\forall I_{i_j} \in I_i \mid I_{i_j}.TBP = compensable \wedge EL_{i_j} \in \{el_k^{-1} \mid el_k \in Log^*, I_k.TBP = compensable\}$;

(2)若 $\forall I_k, I_p \in I_i \wedge I_k \prec I_p$,满足 $I'_p \prec I'_k$;

(3)σ_{i_j} 是可规约补偿的,$\exists el_k, el_k^{-1} \in Log^*$ 且 $I_k \prec$

$I_k^{'}$, $Reduced_{cp}(\sigma_{i_j})$规约 el_k 和 el_k^{-1} 后 $|\sigma_{i_j}|=0$;

(4)若 $el_k, el_k^{-1}\in Log^*$, el_k 和 el_k^{-1} 均是免于失败的,从 σ_{i_j}中去掉 el_k 和 el_k^{-1},规约 σ_{i_j}直到 $|\sigma_{i_j}|=0$,则 σ_{i_j} 免于失败.

3　失败恢复机制

基于组合 Web 事务的综合恢复机制包括向后、向前和替代恢复.分两步骤构建恢复模型:

(1)创建逆库所、逆变迁和逆向流:I 执行失败,需依据正向流构造逆向恢复流.若有 $I=(P,T,F)$,有必要构造逆向恢复网 $I^{-1}=(P^{-1},T^{-1},F^{-1})$,其中:①$P^{-1}=P'$, $p\in P$, p' 是 p 的逆库所;②$T^{-1}=T'$,且 $t\in T$, t' 是 t 的逆变迁;③$F^{-1}\subseteq\{(P'\times T')\mid(T\times P)\subset F\}\cup\{(T'\times P')\mid(P\times T)\subset F\}$;④$\ell^{-1}=\ell\cup\ell^{-1}\cup\{\tau\}$.

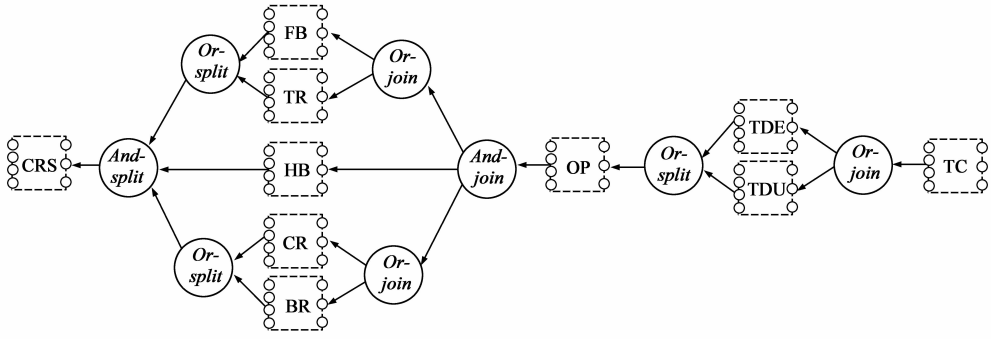

图3　逆向恢复模型

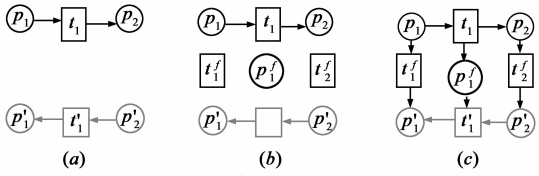

图4　建立恢复映射

(2)建立正向流和逆向恢复流之间映射:在变迁 t_i 和 $t_i^{'}$ 之间创建失败库所 p_i^f;在库所 p_i 和 $p_i^{'}$ 之间创建失败变迁 t_i^f,如图 3(b)所示.若 t_i 失败,则 p_i 激活 t_i^f 到达 $p_i^{'}$.若有正向流 $I=(P,T,F)$,相应逆向恢复网 $I^{-1}=(P^{-1},T^{-1},F^{-1})$,引入失败变迁 T^f 和失败库所 P^f,使得对于任意 $p_i\in P$ 都有唯一的 $t_i^f\in T^f$,对于每个 $t_j\in T$,都有唯一的 $p_j^f\in P^f$.则 I 对应的恢复配对网 $\bar{I}=(\bar{P},\bar{T},\bar{F})$,其中①$\bar{P}=P\cup P^{-1}\cup P^f$;②$\bar{T}=T\cup T^{-1}\cup T^f$;③$\bar{F}=F\cup F^{-1}\cup F^f$,其中 $F^f=\{(p_i,t_i^f)\mid p_i\in P\wedge t_i^f\in T^f\}\cup\{(t_i^f,p_i^{'})\mid t_i^f\in T^f\wedge p_i^{'}\in P'\}\cup\{(t_j,p_j^f)\mid t_j\in T\wedge p_j^f\in P^f\}\cup\{(p_j^f,t_j^{'})\mid p_j^f\in P^f\wedge t_j^{'}\in T'\}$.为了区别正向流,逆向流用灰色表示.

3.1　向后恢复机制

采用扩展Petri网形式化描述时,分两步构建事务

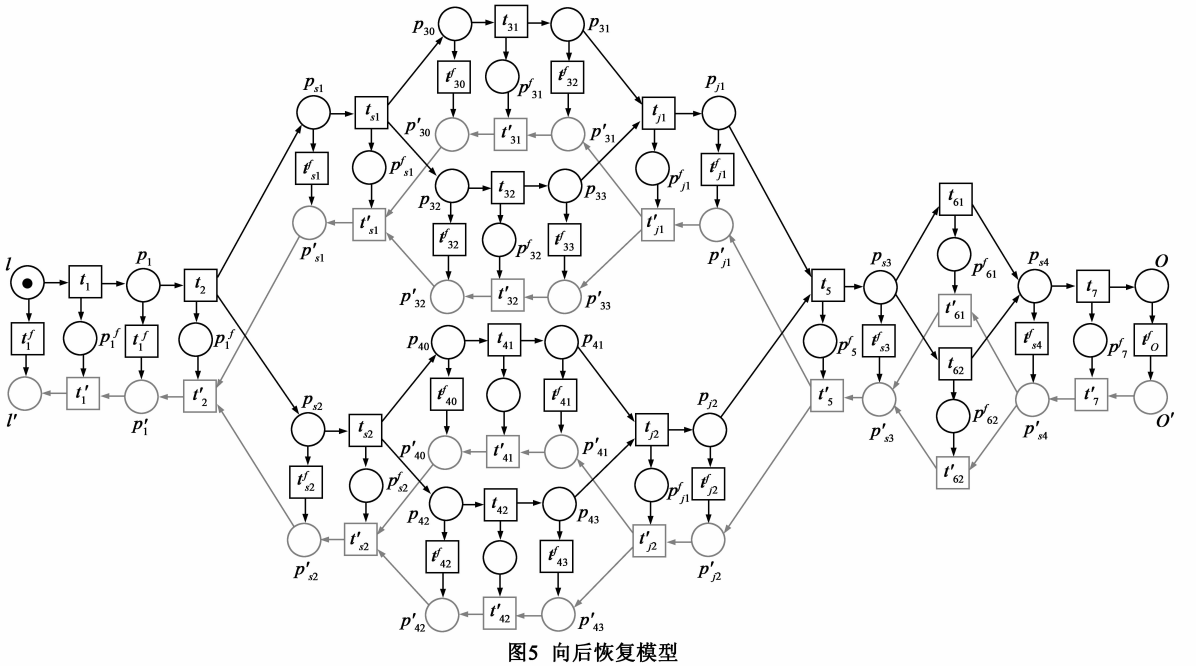

图5　向后恢复模型

流的向后恢复模型:(1)引入恢复库所 P^f 和失败转移变迁 T^f,T^f 将事务流转入恢复流,P^r 描述恢复确认状态;(2)建立 I_i 和 I'_i 之间的映射,T^f 实现正常业务流程到恢复流之间转移,*RH* 激活 T^f,到达恢复流(I_i 转移 I'_i).*failed* 触发 *RH*(Recovery Handler),使控制流转向恢复流.下面引入向后恢复配对策略 BRPS(Backward Recovery Paired Strategy):

业务流程中各 I 及其 I' 的状态/变迁图表示分别为 $I_i=(P_i,T_i,F_i)$ 和 $I'_i=(P'_i,T'_i,F'_i)$,$\exists \Xi \forall I_i \in \Xi \mid I_i \div I'_i \in T_i \div T'_i$,$t_i^c$ 是 I_i 到 I_{i+1} 的转移变迁,t_i^f 是 I_i 到 I'_i 的失败转移变迁,t'^c_i 是 I'_i 到 I'_{i+1} 的补偿转移变迁.*BRPS* 的三元组表示为 $(\bar{P},\bar{T},\bar{F})$,其中:(1)$\bar{P}=P\cup P'$;(2)$\bar{T}=T\cup T'\cup T^f\cup T^c\cup T'^c$;(3)$\bar{F}=F\cup F'\cup F^c\cup F^f\cup F'^c$,其中 $F^c=\{(p_i,t_i^c)\cup(t_i^c,p_{i+1})\mid p_i,p_{i+1}\in P,t_i^c\in T^c\}$,$F^f=\{(p_i,t_i^f)\cup(t_i^f,p'_i)\mid p_i\in P,p'_i\in P',t_i^f\in T^f\}$,$F'^c=\{(p'_{i+1},t_i'^c)\cup(t_i'^c,p'_i)\mid p'_i,p'_{i+1}\in P,t_i'^c\in T'^c\}$,如图 5 所示.

3.2 向前恢复机制

向前恢复机制 FRPS(Forward Recovery Paired Strategy)构造类似于 BRPS,唯一不同的是回滚到特定的终止依赖点 TDP(Terminate Dependency Point)后,RH 修复失败(注入正确参数),重新启动正向流继续执行.存在执行失败任务 I_j,计算其 TDP,确定 Ξ,FRPS 的三元组表示为 $(\overleftrightarrow{P},\overleftrightarrow{T},\overleftrightarrow{F})$ 其中:(1)$\overleftrightarrow{P}=P\cup\bar{P}$;(2)$\overleftrightarrow{T}=T\cup\bar{T}\cup\{t^r\}$;(3)$\overleftrightarrow{F}=F\cup\bar{F}\cup\{(p'_i,t^r),(t^r,p_i)\}$,如图 6 所示.

RH 分三步构建 FRPS:第一步,终止 TCS 流程的执行,确定 I_i 的 TDP 和 Ξ;第二步,执行恢复任务;第三步,修复失败 TDP 输入参数,重新启动 TCS 流程.注意,若 I_i 的 TDP 为 I_1 时,FRPS 和 BRPS 执行恢复上没有区别.

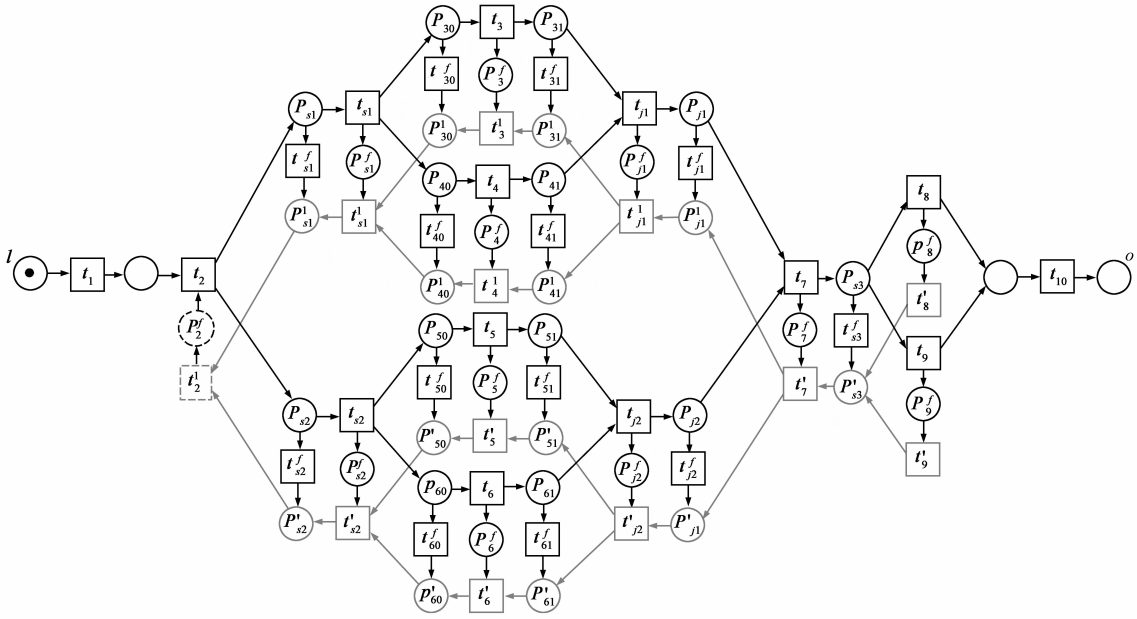

图6 向前恢复模型

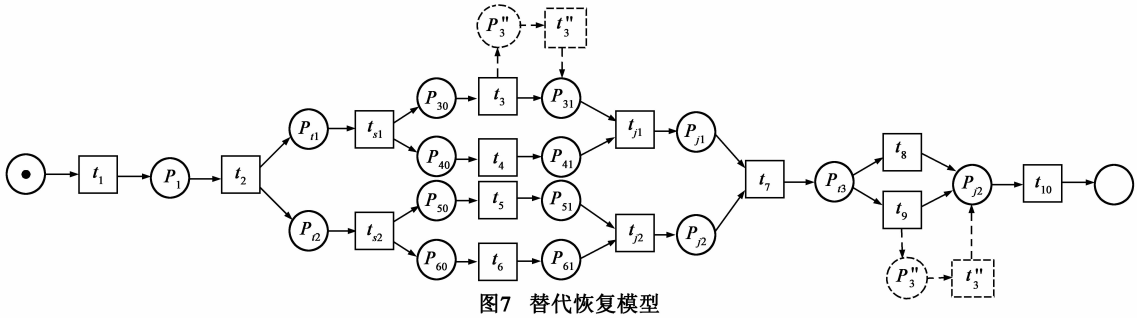

图7 替代恢复模型

3.3 替代恢复机制

替代恢复机制是当执行 I_j 出现失败时,计算其终止依赖点 TDP 为 I_i,确定 Ξ,若满足 $\{I_j\}=\{I_i\}=\Xi$,则判定是由 I_i 本身导致的失败,而不依赖于已提交任务,故不需要执行向后恢复,避免不必要的执行代价.一种简单方法只需触发 I_i 的替代服务 I''_i,点火 I''_i 并执行它.

若执行 TCS 中 I_i 失败,计算其 $TDPI_i$,确定 Ξ,满足 $\exists\Xi\forall I_j\in\Xi\mid I_i=I_j$,增加替代任务 $I''_i=(P''_i,T''_i,F''_i)$和替代状态库所 p'',替代恢复策略 ARS(Alternative Recovery Strategy)的三元组宝石为$(\vec{P},\vec{T},\vec{F})$,其中:(1)$\vec{P}=P\cup P''$;(2)$\vec{T}=T\cup T''\cup\tau$;(3)$\vec{F}=F\cup F''$,如图 7 所示.

4　聚合模式的执行语义分析

讨论聚合模式的执行语义之前,下面先讨论单个任务在各种动作事件发生时可能产生下列行为:

(1)$\zeta_{t_i}\vdash(t_i,\square,\tilde{c}^{es,cont_i})\rightarrow(t_i,\diamondsuit,\tilde{c}^{cmd,cont_{i+1}})$,任务 t_i 成功执行,转入状态 $t_i.Committed$;

(2)$\zeta_{t_i}\vdash(t_i,\square,\tilde{c}^{es,cont_i})\rightarrow(t_i,\triangledown,\tilde{c}^{fal,cont_{i+1}})$,任务 t_i 执行失败,转入状态 $t_i.Failed$;

(3)$\zeta_{t_i}\vdash(t_i,\square,\tilde{c}^{es,cont_i})\rightarrow(t_i,\perp,\tilde{c}^{abt,cont_{i+1}})$,中止任务 t_i 的执行,转入状态 $t_i.Aborted$;

(4)$\zeta_{t_i}\vdash(t_i,\diamondsuit,\tilde{c}^{es,cont_i})\rightarrow(t_i,\lightning,\tilde{c}^{cmp,cont_{i+1}})$,对已经提交的任务 t_i 补偿,转入状态 $t_i.Compensated$;

(5)$\zeta_{t_i}\vdash(t_i,\triangledown,\tilde{c}^{fal,cont_i})\rightarrow(t_i,\triangledown_{\leftarrow},\tilde{c}^{br,cont_{i+1}})$,任务 t_i 执行失败,执行向后恢复;

(6)$\zeta_{t_i}\vdash(t_i,\triangledown,\tilde{c}^{fal,cont_i})\rightarrow(t_i,\triangledown_{\underset{\leftarrow}{\rightarrow}},\tilde{c}^{bfr,cont_{i+1}})$,任务 t_i 执行失败,执行向前恢复;

(7)$\zeta_{t_i}\vdash(t_i,\triangledown,\tilde{c}^{fal,cont_i})\rightarrow(t_i,\triangledown\updownarrow,\tilde{c}^{sr,cont_{i+1}})$,任务 t_i 执行失败,执行替代恢复.

这里$\square\in\{\diamondsuit,\triangledown,\perp,\lightning\}$表示任务执行状态,$\{\triangledown_{\leftarrow},\triangledown_{\underset{\leftarrow}{\rightarrow}},\triangledown\updownarrow\}$是失败任务触发特定恢复机制,$\tilde{c}^{e,cont}$为任务执行日志,$(t,\square,\tilde{c}^{e,cont})$描述任务 t 在特定执行状态的行为语义,可将任务行为语义简记为 $\xi\vdash cfg_1\rightarrow cfg_2$,$cfg_1$ 为开始格局,cfg_2 为完成格局.对于 $I_1\odot I_2\odot\cdots\odot I_n,\odot\in\{\oplus,\|,\otimes,\Theta\}$,从初始格局讨论各聚合分支的执行语义:

$$\begin{cases}\zeta_{t_1}\vdash(t_1,\square,\tilde{c}^{es,cont_1})\\ \cdots\\ \zeta_{t_{i-1}}\vdash(t_{i-1},\square,\tilde{c}^{es,cont_{i-1}})\\ \zeta_{t_i}\vdash(t_i,\square,\tilde{c}^{es,cont_i})\\ \cdots\\ \zeta_{t_n}\vdash(t_n,\square,\tilde{c}^{es,cont_n})\end{cases}\Rightarrow\begin{cases}(t_1,\diamondsuit,\tilde{c}^{cmd,cont_2})\\ \cdots\\ (t_{i-1},\diamondsuit,\tilde{c}^{cmd,cont_i})\\ (t_i,\diamondsuit,\tilde{c}^{cmd,cont_{i+1}})\\ \cdots\\ (t_n,\diamondsuit,\tilde{c}^{cmd,cont_{n+1}})\end{cases}\text{or}$$

$$\begin{cases}(t_1,\triangledown,\tilde{c}^{fal,cont_2})\\ \cdots\\ (t_{i-1},\diamondsuit,\tilde{c}^{cmd,cont_i})\\ (t_i,\diamondsuit,\tilde{c}^{cmd,cont_{i+1}})\\ \cdots\\ (t_n,\diamondsuit,\tilde{c}^{cmd,cont_{n+1}})\end{cases}\text{or}\begin{cases}(t_1,\diamondsuit,\tilde{c}^{cmd,cont_2})\\ \cdots\\ (t_{i-1},\diamondsuit,\tilde{c}^{cmd,cont_i})\\ (t_i,\triangledown,\tilde{c}^{fal,cont_{i+1}})\\ \cdots\\ (t_n,\square,\tilde{c}^{e,cont_{n+1}})\end{cases}$$

$$\begin{cases}(t_1,\diamondsuit,\tilde{c}^{cmd,cont_2})\\ \cdots\\ (t_{i-1},\diamondsuit,\tilde{c}^{cmd,cont_i})\\ (t_i,\triangledown,\tilde{c}^{fal,cont_{i+1}})\\ \cdots\\ (t_n,\square,\tilde{c}^{e,cont_{n+1}})\end{cases}\text{or}\begin{cases}(t_1,\diamondsuit,\tilde{c}^{cmd,cont_2})\\ \cdots\\ (t_{i-1},\diamondsuit,\tilde{c}^{cmd,cont_i})\\ (t_i,\triangledown\updownarrow,\tilde{c}^{fal,cont_{i+1}})\\ \cdots\\ (t_n,\square,\tilde{c}^{e,cont_{n+1}})\end{cases}$$

$\zeta_{t_i}\vdash(t_i,\triangledown,\tilde{c}^{fal,cont_{i+1}})$失败发生,恢复引入到格局中:

$$\begin{cases}(t_1,\diamondsuit,\tilde{c}^{cmd,cont_2})\\ \cdots\\ (t_{i-1},\diamondsuit,\tilde{c}^{cmd,cont_i})\\ (t_i,\triangledown_{\leftarrow},\tilde{c}^{fal,cont_{i+1}})\\ \cdots\\ (t_n,\square,\tilde{c}^{e,cont_{n+1}})\end{cases},\begin{cases}(t_1,\diamondsuit,\tilde{c}^{cmd,cont_2})\\ \cdots\\ (t_{i-1},\diamondsuit,\tilde{c}^{cmd,cont_i})\\ (t_i,\triangledown_{\underset{\leftarrow}{\rightarrow}},\tilde{c}^{fal,cont_{i+1}})\\ \cdots\\ (t_n,\square,\tilde{c}^{e,cont_{n+1}})\end{cases}$$

$$\text{or}\begin{cases}(t_1,\diamondsuit,\tilde{c}^{cmd,cont_2})\\ \cdots\\ (t_{i-1},\diamondsuit,\tilde{c}^{cmd,cont_i})\\ (t_i,\triangledown\updownarrow,\tilde{c}^{fal,cont_{i+1}})\\ \cdots\\ (t_n,\square,\tilde{c}^{e,cont_{n+1}})\end{cases}$$

下面讨论顺序、并行、选择、鉴别器和迭代聚合的失败恢复:

$$\begin{pmatrix}(trigger(I_1,invoke^{RH}_{I_1})\wedge getCpst(I_1,r_{I_{1,\oplus}},\oplus_{I_1},\tilde{c}^{e,\alpha})\\ \wedge CHbegin(I'_1,HB_{\oplus_{I_1}}))\wedge\\ \cdots\\ \wedge(trigger(I_i,invoke^{RH}_{I_i})\wedge getCpst(I_i,r_{I_{i,\oplus}},\oplus_{I_i},\tilde{c}^{e,\alpha})\\ \wedge CHbegin(I'_i,HB_{\oplus_{I_i}}))\wedge\\ \cdots\\ \wedge(trigger(I_n,invoke^{RH}_{I_n})\wedge getCpst(I_n,r_{I_{n,\oplus}},\oplus_{I_n},\tilde{c}^{e,\alpha})\\ \wedge CHbegin(I'_n,HB_{\oplus_{I_n}}))\end{pmatrix}$$

假定处于 *initial* 或 *final* 的顺序聚合任务 $\oplus^e$ 执行分别描述为和 $\oplus^e_f=\{(\oplus^e_b,f_{I_i,oy})\mid I_i\in\oplus^e\wedge oy\in out(I_i)\}$,处于 *initial* 或 *final* 的顺序聚合恢复任务 $\oplus^r$ 执行分别描述为 $\oplus^r_r=\{(r_{I_i,iy},\oplus^r_b)\mid I'_i\in\oplus^r\wedge iy\in out(I'_i)\}$ 和 $\oplus^f_r=\{(\oplus^r_b,f'_{I_i,oy})\mid I'_i\in\oplus^r\wedge oy\in in(I'_i)\}$.其中,$trigger(I_i,invoke^{RH}_{I_i})\Rightarrow(I_{ci},\oplus_{Seq})$,$I_i$ 触发 *RH* 并由 $getOp(I'_i,\oplus_{Sequence})$? $OP_{I'_i}:putprt\oplus_{Seq}$获取$\oplus_{Seq}$的恢复任务$I'_i$,其中 $OP_{I'_i}$获取 I_{Seq}的恢复任务,否则 $putprt\oplus_{Seq}$从其父范围取 $OP_{I'_i}$;$getCpst(I_i,r_{I_i},\oplus_{I_i},\tilde{c}^{e,\alpha})$启动 $binding(I_i,I'_i,r_{I'_i},\tilde{c}^{e,\alpha})$建立 I_i 和 I'_i 之间映射,其中 $I_i\cap I'_i=\varnothing$ 且 $I_i\times$

$P^{I_i} \times I_i^{'}$; $CHbegin(I_i^{'}, HB_{S'I_i}) \Rightarrow rests(I_i^{'}, HB_{S'I_i})$,返回恢复状态.

若顺序聚合中任务集是 compensable 或 nonvital,使所有元素失败,中断顺序聚合执行,但对整个 LRTs 没有任何影响,因为 nonvital 任务对 LRTs 成功提交不是关键的.假定 I_i 尝试用不同 WS 执行多次,启用前向恢复,用不同 WS 重试,这时需满足如下条件:① $I_i.state = Executing$;② $I_i.TBP = Vital$.本文不讨论补偿再次失败问题,这仍是一个悬而未决的复杂问题.下面分情形讨论其执行语义:

(1) $\zeta_{I_1 \oplus \cdots \oplus I_n} \rightarrow (I_1 \oplus \cdots \oplus I_n, \square, \tilde{c}^{es, cont_1}) \rightarrow (I_1 \oplus \cdots \oplus I_n, \diamondsuit, \tilde{c}^{cmd, cont_{n+1}})$,不论 $I_i.TBP(1 \leqslant i \leqslant n)$是哪种原子事务类型,只要每一任务 $I_i.Committed$ 均成功提交,则 $I_1 \oplus I_2 \oplus \cdots \oplus I_n$ 成功提交,继续执行.

(2) $\zeta_{I_1 \oplus \cdots \oplus I_n} \rightarrow (I_1 \oplus \cdots \oplus I_n, \nabla, \tilde{c}^{fal, cont_1}) \rightarrow (I_1 \oplus \cdots \oplus I_n, \nabla_{\perp I_1}, \tilde{c}^{br, cont_2})$,引擎捕获 I_1 状态 *Failed*,激活 *RH*,存在三种恢复:①若 $TBP(I_1)$为 *Retriable* 且 S 为 $\varnothing$,则启动 *ARS*;②若 S 不为 $\varnothing$,且存在执行迹 σ,满足$(I, Committed) \in \sigma \wedge I.TBP \in \{Vital, Compensable\}$,启动 *BRPS*;③修复失败发生,启动 *FRPS*;

(3) $\zeta_{I_1 \oplus \cdots \oplus I_n} \rightarrow (I_1 \oplus \cdots \oplus I_n, \nabla, \bigcup_{k=1}^{i-1} \tilde{c}^{cmd, cont_k} \cup \tilde{c}^{fal, cont_i}) \rightarrow (I_1 \oplus \cdots \oplus I_n, \nabla_{\perp I_1}, \bigcup_{k=1}^{i-1} \tilde{c}^{cmd, cont_k} \cup \tilde{c}^{br, cont_{i+1}})$,其中 $I_1, \cdots, I_{i-1}$均已成功提交,且执行迹满足 $\forall I_k | (I_k, committed) \in \sigma_1 \wedge 1 \leqslant k \leqslant |\sigma_1|$,由 $I_1 \oplus \cdots \oplus I_n$ 由 $\sigma_0\sigma_1$ 组成,σ_0 和 σ_1 可以按情形(2)讨论三种恢复方式分别处理,不同的是$(I_1 \oplus \cdots I_n, \nabla \tilde{c}^{fal, cont_i},) \rightarrow (I_1 \oplus \cdots \oplus I_n, \nabla_{I_1 \cdots I_{i-1} \rightarrow I_i}, \tilde{c}^{br, cont_{i+1}})$.

(4) $\zeta_{I_1 \oplus \cdots \oplus I_n} \rightarrow (I_1 \oplus \cdots \oplus I_n, \perp_{I_i}, \bigcup_{k=1}^{i-1} \tilde{c}^{cmd, cont_k} \cup \tilde{c}^{abt, cont_i})$,其中 $I_1, \cdots, I_{i-1}$均为 *Committed*,而 I_i 的执行状态为 $I_i.runnung$,当外部事件 *Abort*()点火 I_i,若存在中止依赖,则已成功执行任务 $I_l \in \{I_1, \cdots, I_{i-1}\}$(其中 $1 \leqslant l \leqslant i-1$)并且 $I_l.TBP \in \{Compensable, Vital\}$,并启动 *BRPS* 执行恢复.

$$
\left(
\begin{array}{c}
getAjh(r_{I_0, ay}, As \parallel_{I_0}^{s}) \wedge \\
\left(
\begin{array}{c}
(trigger(I_1, invoke_{I_1}^{RH}) \wedge getCpst(I_1, r_{I_1, As}, S_{I_1}, \tilde{c}^{e,\alpha}) \\
\wedge CHbegin(I_1^{'}, HB_{S_{A1}})) \wedge \\
\cdots \\
\wedge (trigger(I_i, invoke_{I_i}^{RH}) \wedge getCpst(I_i, r_{I_i, As}, S_{I_i}, \tilde{c}^{e,\alpha}) \\
\wedge CHbegin(I_i^{'}, HB_{S_{I_i}}) \wedge \\
\cdots \\
\wedge (trigger(I_n, invoke_{I_n}^{RH}) \wedge getCpst(A_n, r_{I_n, As}, S_{I_n}, \tilde{c}^{e,\alpha}) \\
\wedge CHbegin(I_n^{'}, HB_{S_{I_n}}))
\end{array}
\right) \\
\wedge getAjt(Aj \parallel_{I_{n+1}}^{e}, r_{I_{n+1}, aw})
\end{array}
\right)
$$

AND-split 将控制流分发到两个或多个并发任务中.处于 *initial* 和 *final* 的 *AND-split* 中任务 $As \parallel^{e}$ 分别描述为 $As \parallel_r^e = \{(f_{I_i, iy}, \parallel^e) | I_i \in \parallel^e \wedge iy \in in(I_i)\}$或 $As \parallel_f^e = \{(As \parallel^e, f_{I_i, oy}) | I_i \in As \parallel^e \wedge oy \in out(I_i)\}$,处于 *initial* 或 *final* 的 *AND-split* 中恢复任务 $As \parallel^r$ 分别描述为 $As \parallel_r^r = \{(r_{I_i, iy}, As \parallel_r) \mid I_i \in As \parallel^r \wedge iy \in out(I_i)\}$或 $As \parallel_f^r = \{(As \parallel^r, f_{I_i, oy}) | I_i \in As \parallel^r \wedge oy$.其中,$getAjh(r_{I_i, iy}, As \parallel_{I_0}^{s}) \Rightarrow (As \parallel_{I_0}^{j}, f_{Ii, ay})$, $trigger(I_i, invoke_{I_i}^{RH}) \Rightarrow (I_i^{'}, S_{Ipar})$,映射恢复事件到 I_{par}的范围,并取恢复任务 $I_i^{'}$,如果无恢复任务执行,则返回 $NonOP_{Slpar}$,否则返回 OP_{Slpar}; $getCpst(I_i, r_{I_i}, S_{I_i, As}, \tilde{c}^{e,\alpha}) \Rightarrow blinding(I_i^{'}, r_{I_i}, S_{Ai, As}, \tilde{c}^{e,\alpha})$,其中 $I_i \cap I_i^{'} = \varnothing$ 且 $I_i \times P^{I_i} \times I_i^{'}$; $CHbegin(I_i^{'}, HB_{S_{I_i}})(retst(I_i^{'}, HB_{S_{I_i}})$,其中$\{(r_{I_i, ay}, RH_{r_{I_0}^c}) | I_i \in I_i^{'}, ay \in in(I_i)\}$返回恢复状态; $getAjt(Aj \parallel_{I_{n+1}}^{s}, r_{I_{n+1}, aw}) \Rightarrow (f_{I_{n+1}, aw}, Aj \parallel_{I_{n+1}}^{f})$,其中$(r_{I_{n+1}, ay}, Aj \parallel_{I_{n+1}}^{s})$表明 *AND-join* 就绪,$(Aj \parallel_{I_{n+1}}^{f}, f_{I_{n+1}, ay})$表明 *AND-join* 完成.

激活$(I_1 \parallel \cdots \parallel I_n)_{succ}$执行,确定 $I_i.state$ 是否为 *Committed*,其是 *vital* 的;否则在不影响执行进展的情况下,可能在其它状态上终止.若$(I_1 \parallel \cdots \parallel I_n)_{succ}$包含一个或几个不可补偿 *vital* 任务,在该模式下提交,一种特殊的同步机制加入并通知该模式中其他不可补偿 *vital* 任务,或者全部提交或无不可补偿 *vital* 任务提交.提交依赖于不可补偿 *vital* 任务的执行进展. $I_1 \parallel \cdots \parallel I_n$ 任务集尝试提交,当且仅当满足 $I_i.behavior = non\text{-}compensatable \wedge I_i.TBP = vital \wedge I_i.state = Committed$.

若$(I_1 \parallel \cdots \parallel I_n)_{succ}$不包含任何 *vital* 任务,$I_m \in (I_1 \parallel \cdots \parallel I_n)_{succ} \wedge I_m.state = Failed$,触发 *FRPS*;若$(I_1 \parallel \cdots \parallel I_n)_{succ}$中所有任务都是 *nonvital*,即使所有元素失败,失败也不影响流程执行,由于 *nonvital* 任务提交对 *LRTs* 不是至关重要的.若尝试重试 *vital* 任务,重试后仍失败.若中断依赖,满足 $I_m \in (I_1 \parallel \cdots \parallel I_n)_{succ} \wedge I_m.state = Executing \wedge I_m.TBP = vital$;若补偿依赖,满足 $I_i \in I_1 \parallel \cdots \parallel I_n \wedge I_i.behavior = compensatable \wedge I_i.state = Committed \wedge I_i.TBP = vital$.若 *LRTs* 包含并行聚合,$I_i$ 是不可补偿的,可行的解决方法是在 I_i 和并行执行任务 $I_1 \parallel \cdots \parallel I_n$ 之间插入一个确认.下面讨论并行聚合模式的执行语义分析:

(1) $\zeta_{I_1 \parallel \cdots \parallel I_n} \rightarrow (I_1 \parallel \cdots \parallel I_n, \square, c^{\sim e, cont_1}) \rightarrow (I_1 \parallel \cdots \parallel I_n, \diamondsuit, \bigcup_{i=1}^{n} c^{\sim cmd, cont_i})$,不论 $I_i.TBP(1 \leqslant i \leqslant n)$是哪种类型,只要 $I_1, \cdots, I_n$ 均提交,则 $I_1 \parallel I_2 \parallel \cdots \parallel I_n$ 成功提交,执行继续.

(2) $\zeta_{I_1 \parallel \cdots \parallel I_n} \rightarrow (I_1 \parallel \cdots \parallel I_n, \nabla, c^{\sim fal, cont_i}) \rightarrow (I_1 \oplus \cdots \oplus I_n, \nabla_{\perp I_i}, c^{\sim br, cont_{i+1}})$,若 $I_i.state$ 为 *Failed*(注意

这里不讨论多个失败同时发生),计算 I_i 失败 *TDP* 并确定 S,激活 *RH*,分别讨论三种恢复:①若 $TBP(I_i)$为 *Retriable* 且 S 为 $\varnothing$,则启动 *ARS*,并存在格局转换 $\zeta_{I_i}\rightarrow(I_i,\nabla,\tilde{c}^{fal,cont_i})\Rightarrow(I_i'',\diamondsuit,\tilde{c}^{cmd,cont_i'})$;② *RH* 向 *Running* 任务发 *Abort*(),若 S 不为 $\varnothing$,且有执行迹 σ,满足 $(I_i,Committed)\in\sigma\wedge I.TBP\in\{Compensable,Vital\}$,进一步形成新迹 σI_i,依据执行依赖和失败契约,启动 *BRPS*,执行格局 $(I_1\parallel\cdots\parallel I_n,\nabla,\tilde{c}^{fal,cont_i})\rightarrow(I_1\parallel\cdots\parallel I_n,\nabla_{\{t_1\cdots t_n\}\backslash I_i\rightarrow t_i},\tilde{c}^{br,cont_i}))$,流程执行继续;③类似(2),依据规则 1 的执行依赖和失败契约,启动 *FRPS*;

(3) $\zeta_{I_1\parallel\cdots\parallel I_n}\rightarrow(I_1\parallel\cdots\parallel I_n,\perp_{I_i},c^{\sim abt,cont_i})$, *RH* 接收 *Abort*(),中止任务执行,触发 *BRPS*,补偿已提交任务. $I_1\parallel\cdots\parallel I_n$ 中 I_i 失败,可能导致不一致状态,满足 $\forall I_k\mid I_k\in\{I_1,\cdots,I_n\}\wedge I_k.TBP\in\{Compensable,Vital\}$ 是一致的;若 $\exists I_k\mid I_k\in\{I_1,\cdots,I_n\}\wedge I_k.TBP\in\{Retriable,Pivot\}$,一旦 I_k 提交,则无法消除其产生的影响,导致执行的不一致性.

$$\left(\begin{array}{c} getOjh(r_{I_0,ay},Os\otimes^s_{I_0})\\ \wedge evalbool(in(I_0))\wedge skip(tokenbl(in(I_0),sel))\wedge\\ \left(\begin{array}{c}(trigger(I_1,invoke^{RH}_{I_1})\wedge getCpst(I_1,r_{I_1,OS},S_{I_1},\tilde{c}^{e,\alpha})\\ \wedge CHbegin(I_1',HB_{S_{I_1}}))\vee\\ \cdots\\ \vee(trigger(I_n,invoke^{RH}_{An})\wedge getCpst(I_n,r_{I_n,OS},S_{I_n},\tilde{c}^{e,\alpha})\\ \wedge CHbegin(I_n',HB_{S_{I_n}}))\end{array}\right)\\ \wedge getOjt(Oj\otimes^e_{I_{n+1}},r_{n+1,aw})\end{array}\right)$$

处于 *initial* 或 *final* 的 *OR-join* 中任务 $Aj\otimes^e$ 分别描述为 $Oj\otimes^e_r=\{(r_{I_i,iy},Oj\otimes^e_{oz})\mid I_i\in Oj\otimes^e\wedge iy\in in(I_i)\wedge oz\in out(I_i)\}$或 $Oj\otimes^e_f=\{(Oj\otimes^e_{I_i},f_{I_i,oy})\mid I_i\in Oj\otimes^e\wedge oy\in out(I_i)\}$,处于 *initial* 或 *final* 的 *OR-join* 中恢复任务 $Oj\otimes^r$ 分别描述为 $Oj\otimes^r_r=\{(r_{I_i,iy},Oj\otimes^r_{oz})\mid I_i\in Oj\otimes^r\wedge iy\in out(I_i)\wedge oz\in out(I_i)\}$或 $Oj\otimes^r_f=\{(Oj\otimes^r_{I_i},f_{I_i,oy})\mid ax\in Oj\otimes^r\wedge oy\in in(I_i)\}$. $precond_{Os\otimes I_0}\wedge getOjh(r_{I_0,iy},Os\otimes^s_{I_i})\Rightarrow(Os\otimes^f_{I_i},f_{I_i,ay})$,其中 $precond_{Os\otimes I_0}$是前提条件; *skip*(*tokenbl*(*in*(I_0),*sel*)), *OR-split* 依据 *evalbool*(*in*(I_0))执行映射 $Os\otimes^f_{I_0}\times I_i\times Aj\otimes^s_{I_{n+1}}$,"真"托肯传递到传递到执行分支,"假"托肯分发到其余分支,并执行 $skip_{I_i}$; *trigger*(I_i,$invoker^{RH}_{I_i}$)⊗(I_i',S_{Isel}),获取 I_i 的 S_{Isel},取补偿操作 $getOp(I_i',S_{Isel})\otimes OP_{I_i'}:putprt_{SIsel}$,取 I_i' 的恢复操作 $OP_{I_i'}$,否则抛到其父范围; $getCpst(I_i,r_{I_i},S_{I_i,As\otimes},\tilde{c}^{e,\alpha})$绑定恢复任务 $binding(I_i,I_i',r_{I_i'},\tilde{c}^{e,\alpha})$,其中 $I_i\cap I_i'=\varnothing$ 且 $I_i\times P^{I_0}\times I_i'$; $CHbegin(I_i,HB_{S_{I_i}})(retsts(I_i,HB_{S_{I_i}})$,返回恢复成功与否状态; $getOjt(Oj\otimes^s_{I_{n+1}},r_{I_{n+1},aw})(f_{I_{n+1},aw},Oj\otimes^f_{I_{n+1}})$,其中 *OR-join* 准备就绪($r_{I_{n+1},ay}$, $OJ^s_{I_{n+1}}$), *AND-join* 完成($Oj\otimes^f_{I_{n+1}},f_{I_{n+1},ay}$).

若 I_m 是$(I_1\otimes I_2\otimes\cdots\otimes I_n)_{pre}$直接前驱, I_m 提交失败,则不激活 $I_1\otimes I_2\otimes\cdots\otimes I_n$.假定 $I_i\in I_1\otimes I_2\otimes\cdots\otimes I_n$ 选择分支,由于 I_i 可能是 *vital* 或 *nonvital* 的,不同于 I_i 的激活条件, *vital* 成功终止处于提交状态,而 *nonvital* 任务可以是其它状态.若 $I_1\otimes I_2\otimes\cdots\otimes I_n$ 至少有一个任务是 *vital* 的,满足 $I_i.TBP=vital$;若 I_i 有效激活,满足 $I_i.TBP=vital\wedge I_m.state=committed$.如果提交失败,需触发向后恢复;激活失败,需要向后恢复.特别地若 I_i 是不可补偿的,则可能导致不一致性.下面讨论不同选择条件下聚合执行语义:

(1) $\zeta_{I_i}\otimes\cdots\otimes I_n^{cond(I_i)}\mid\rightarrow(I_1\otimes\cdots\otimes I_n,\square,\tilde{c}^{es,cont_1})$,若 $\zeta_{I_i}\otimes\cdots\otimes I_n\mid^{cond(I_i)=true}$,分两种情形讨论:(ⅰ)若$(t_i,\square,\tilde{c}^{es,cont_1})^{true}\rightarrow(t_i,\nabla,\tilde{c}^{fal,cont_i})^{true}$,引擎获取 I_i 的执行状态 *Failed*,计算 I_i 的 *TDP* 并确定恢复范围 S,启动 *RH*,有三种恢复方式:①若 $I_i.TBP$ 为 *Retriable* 且 S 仅涉及一个任务,则启动 *ARS* 恢复,并有格局转换$(I_i,\nabla,\tilde{c}^{fal,cont_i})^{true}\rightarrow(I_i'',\diamondsuit,\tilde{c}^{cmd,cont_i'})^{true}$;②若 S 不为□,且对应执行迹为 σ,满足 $(I,Committed)\in\sigma\wedge I.TBP\in\{Compensable,Vital\}$,则有迹 σI_i,依据执行依赖和恢复契约,启动 *BRPS* 执行逆向补偿$(I_i,\nabla,\tilde{c}^{fal,cont_i})^{true}\rightarrow(I_i,\nabla_{\perp Ii},\tilde{c}^{br,cont_i})^{true}$,执行继续;③类似②,启动 *FRPS*;(ⅱ)若$(I_1\otimes\cdots\otimes I_n,\square,\tilde{c}^{es,cont_i})\rightarrow(I_i,\diamondsuit,\tilde{c}^{cmd,cont_{i+1}})$,引擎获取 I_i 的执行状态 *Committed*,继续向后执行.

(2) 对于 $\zeta_{I_1\otimes\cdots\otimes I_n}\mid^{cond(I_1)\vee\cdots\vee cond(I_n)=false}$,计算选择条件 $case(cond_1)\vee\cdots\vee case(cond_n)$均为 *false*,没有任何分支被激活,绕过 $I_1\otimes I_2\otimes\cdots\otimes I_n$ 继续执行后续流程.

$$\left(\begin{array}{c} getAjh(r_{I_0,ay},As\parallel^s_{I_0})\wedge\\ \left(\begin{array}{c}(trigger(I_1,invoke^{RH}_{I_1})\wedge getCpst(I_1,r_{I_1,As},S_{I_1},\tilde{c}^{e,\alpha})\\ \wedge CHbegin(I_1',HB_{S_{A1}}))\wedge\\ \cdots\\ \wedge(trigger(I_i,invoke^{RH}_{I_i})\wedge getCpst(I_i,r_{I_i,As},S_{I_i},\tilde{c}^{e,\alpha})\\ \wedge CHbegin(I_i',HB_{S_{I_i}})\wedge\\ \cdots\\ \wedge(trigger(I_n,invoke^{RH}_{I_n})\wedge getCpst(A_n,r_{I_n,As},S_{I_n},\tilde{c}^{e,\alpha})\\ \wedge CHbegin(I_n',HB_{S_{I_n}}))\end{array}\right)\\ \wedge guard(status)\wedge getOjt(Oj\otimes^e_{I_{n+1}},r_{n+1,aw})\end{array}\right)$$

结合了 *AND-split* 和 *OR-join* 聚合模式.仅需在 I_1,

$I_2,\cdots,I_n$ 执行分支后加入守护函数 *guard*(*status*),其中 $status\in\{\diamond,\triangledown,\perp\}$,通过 $guard(t_i)\wedge\zeta_{I_1\Theta\cdots\Theta I_n}\rightarrow(I_1\Theta\cdots\Theta I_n,\square,\tilde{c}^{es,cont_i})$捕获最先提交的分支,其执行语义结合了并行和选择聚合模式的执行语义,详细分析略.

$*(trigger(I_i,invoke_{I_i}^{RH})\wedge getCpst(I_i,r_{I_i,OS},S_{I_i},\tilde{c}^{e,\alpha})\wedge Ch\ begin(I_i^{'},HB_{S_{I_i}}))$

$^{cond}\circledS$ 可分解为 $skip\bigcap I_1\bigcap(I_1\oplus I_1)\bigcap(I_1\oplus I_1\oplus I_1)\bigcap\cdots$,选择分支(*sel*,$\perp$,$J$,$K$)$_O^I$,$\ulcorner J\urcorner^I$ 和 $\ulcorner K\urcorner_O$ 的迭代 I/O 接口分别为 $iy_{I_1}^1,iy_{I_1}^2,\cdots,iy_{I_1}^m$ 和 $oy_{I_1}^1,oy_{I_1}^2,\cdots,oy_{I_1}^m$,$m$ 为迭代次数.假定 $^{cond}\circledS$ 执行动作描述为 $skip\oplus\perp$,必存在 $skipU\ulcorner I_1\urcorner_O^I\cup\ulcorner I_1\oplus I_1\urcorner_O^I\cup\ulcorner I_1\oplus I_1\oplus I_1\urcorner_O^I\cdots,trigger(I_i,invoke_{I_1}^{RH})\Rightarrow(I_1^{'},S_{lite})$,取 I_1 的 S_{lite};$getOp(I_1^{'},S_{lite})?Op_{I_1}^{'}:putprt_{Slite}$取 $I_1^{'}$ 的恢复操作 $Op_{I_1}^{'}$,否则抛向其父范围;$getCpst(I_1,r_{I_1},S_{I_1},\tilde{c}^{e,\alpha})$绑定恢复任务,执行 $binding(I_1,I_1^{'},r_{I_1}^{'},\tilde{c}^{e,\alpha})$,其中 $I_1\bigcap I_1^{'}=\varnothing$ 和 $I_1\times P^{I_{lite}}\times I_1^{'}$;$CH\ begin(I_1^{'},HB_{lite})\Rightarrow(retsts(I_1^{'},S_{lite})$返回迭代恢复结果.

迭代聚合模式中任务 I_1 反复执行 λ 次,迭代次数依赖于执行语义,其是一种特殊的选择聚合模式,不同的是仅包含一个元素.下面讨论执行语义:

(1)对于 $^{cont}\circledS\vdash(\perp,\square,\tilde{c}^{e,cont_1})^{cond}\rightarrow(\perp,\diamond,\tilde{c}^{cmd,cont_2})^{cond}$,如果 *cond* 为 *true*,若 I_1 成功提交并到达状态 $I_1.Committed$,则继续执行迭代,直到 *cond* 为 *false*,即迭代完毕.

(2)对于 $^{cond}\circledS\vdash(\perp,\square,\tilde{c}^{e,cont_1})^{cond}\rightarrow(\perp,\triangledown,\tilde{c}^{fal,cont_2})^{cond}$,若 *cond* 为 *true*,而 I_1 转到状态 $I_1.Failed$,若 $I_1.TBP\in\{Vital,Compensable\}$,参照顺序聚合模式处理失败,激活恢复策略;若 $I_1.TBP\in\{Retriable\}$,重试该服务;若 $I_1.TBP\in\{Pivot\}$,失败可能导致不一致性.

5 基于组合事务的应用

本文的研究成果已应用到旅行社的旅行流程安排中,该流程执行涉及酒店、航空公司和汽车租赁,旅行社在广州,酒店、航空公司和汽车租赁公司分布在全国 20 多个旅行目的地,表 1 给出了 TRP 中各被组合服务及其事务性质.旅行社负责下达旅行计划,酒店、航空公司和汽车租赁公司分别执行房间预订、机票预订和租赁汽车,服务提供方接受订单,组织预订并及时反馈执行结果.组合事务的执行语义受各种因素的影响,不同任务对组合事务成功执行作用是不一样的,失败恢复可以动态调整 Web 服务.

表 1 可用服务及其事务性质

Task	*WS*	*TP*	*Task*	*WS*	*TP*	*Task*	*WS*	*TP*	*Task*	*WS*	*TP*	*Task*	*WS*	*TP*
	$WS_{0,1}$	*p*		$WS_{1,1}$	*cp*		$WS_{2,1}$	*cp*		$WS_{3,1}$	*r*		$WS_{4,1}$	*r*, *cp*
	$WS_{0,2}$	*r*		$WS_{1,2}$	*r*		$WS_{2,2}$	*p*		$WS_{3,2}$	*r*, *cp*		$WS_{4,2}$	*p*
	$WS_{0,3}$	*r*		$WS_{1,3}$	*r*, *cp*		$WS_{2,3}$	*cp*		$WS_{3,3}$	*p*		$WS_{4,3}$	*r*, *cp*
CRS	$WS_{0,4}$	*p*	*FB*	$WS_{1,4}$	*p*	*TR*	$WS_{2,4}$	*r*	*HB*	$WS_{3,4}$	*cp*	*CR*	$WS_{4,4}$	*cp*
	$WS_{0,5}$	*r*		$WS_{1,5}$	*p*		$WS_{2,5}$	*r*, *cp*		$WS_{3,5}$	*p*		$WS_{4,5}$	*p*
	$WS_{0,6}$	*p*		$WS_{1,6}$	*r*, *cp*		$WS_{2,6}$	*p*		$WS_{3,6}$	*cp*		$WS_{4,6}$	*cp*
	$WS_{0,7}$	*p*		$WS_{1,7}$	*r*		$WS_{2,7}$	*r*, *cp*		$WS_{3,7}$	*cp*			
	$WS_{5,1}$	*p*		$WS_{6,1}$	*p*		$WS_{7,1}$	*r*		$WS_{8,1}$	*p*		$WS_{9,1}$	*r*
	$WS_{5,2}$	*r*, *cp*		$WS_{6,2}$	*r*		$WS_{7,2}$	*p*		$WS_{8,2}$	*cp*		$WS_{9,2}$	*p*
BR	$WS_{5,3}$	*cp*	*OP*	$WS_{6,3}$	*r*	*TDE*	$WS_{7,3}$	*p*	*TDU*	$WS_{8,3}$	*cp*	*TC*	$WS_{9,3}$	*p*
	$WS_{5,4}$	*cp*		$WS_{6,4}$	*p*		$WS_{7,4}$	*r*		$WS_{8,4}$	*cp*		$WS_{9,4}$	*r*
	$WS_{5,5}$	*r*, *cp*								$WS_{8,5}$	*p*			
	$WS_{5,6}$	*p*								$WS_{8,6}$	*p*			

我们在 Active Endpoints 公司提供的 ActiveBPEL 引擎[13]上扩展组合事务恢复策略,试验环境配置:2 台 IBM x3650 服务器(4 核 Intel 2930MHz 处理器、1GB 内存和 SuSE 10.2 Linux 操作系统)和 12 台网络终端(Intel Pentium E5200 2.5 GHz,2GB RAM 和 Windows XP SP2).除了沿用执行引擎、流程编排器和事件处理器等组件外,扩展恢复处理器和依赖规则库,增加执行日志库 CSFLog,便于重构恢复上下文,形成恢复流.执行恢复时,ActiveBPEL 恢复引擎需重构恢复执行序列、任务执行依赖和外部事件交互.将扩展 CTL 逻辑引入到 TRPFlow 中构造语义执行验证工具 TCS-Tools,在 TCS-Tools 上执行 TRP 事务来验证器执行语义的正确性,图 8 给出聚合 TRP 结构及其依赖语义,在 TCS-Tools 上执行 TRP 事务,依据预定义的失败事务恢复模型和聚合模式的执行语义,验证 TRP 事务执行性质.图 9 所示组合事务的执行语义保证组合事务正确执行.其不仅可以检测 TRP 执行的逻辑错误,还能检测执行失败恢复的正确性.下面讨论 TRP 执行语义:

(1)TRP 成功执行,即 ActiveBPEL 执行 CR 或 BR 后,获取 CSFLog 日志,加载恢复上下文 $\tilde{c}^s$,为 TRP 构造恢复策略.失败发生时,触发恢复处理器.

(2)TRP 执行失败,需启动恢复策略 $CH(A,C,\tilde{c}^s,E_n)$,其中 A 为失败任务,C 为 A 的恢复任务,$\tilde{c}^s$ 从 CSFLog 获取的恢复上下文,E_n 触发恢复事件,可能产生三种情形:

(ⅰ)$CRS\oplus Booking\&Renting((FB\otimes TR)\parallel HB\parallel(CR\otimes BR))$执行失败,依据顺序聚合的失败恢复执行

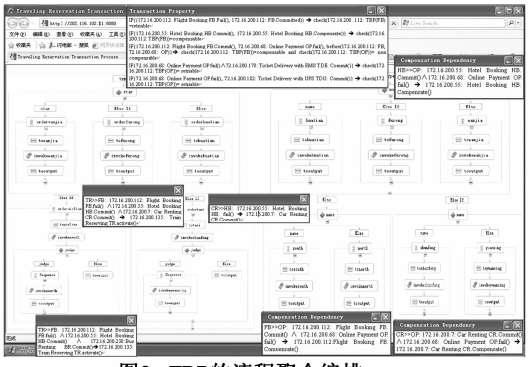

图8　TRP的流程聚合编排

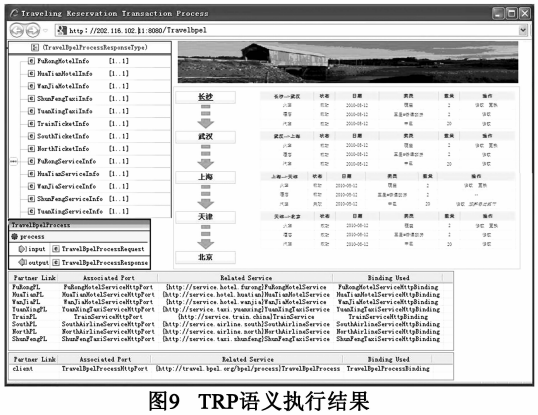

图9　TRP语义执行结果

语义,加载恢复上下文 $\tilde{c}^s$. *RH* 构建 *CRS*, *Booking&Renting* 和相应恢复任务的映射,由于 $RH(A,\varnothing,\tilde{c}^s,E_n)$均为 τ(空操作),故无需恢复.

(ⅱ)$(FB\otimes TR)\parallel HB\parallel(CR\otimes BR)$失败,即 *FB*(或 *TR*)、*HB* 和 *CR*(或 *BR*)之一失败,加载恢复策略,存在如下恢复情形:①执行恢复,依赖并行聚合中子任务:(*a*)*RH* 立即中止子任务执行,安装 $RH((FB\otimes TR)\parallel HB\parallel(CR\otimes BR),((FB'\otimes\varnothing)\parallel HB')\parallel(CR'\otimes BR'))$, $\tilde{c}^{s,par}$, *compnst _ trigger*),由于 $\tilde{c}^{s,par}$非空,执行 *FRPS*;(*b*)当并行子任务均成功执行,安装 *RH*,依次构建恢复上下文 $\tilde{c}^{s,FB}$, $\tilde{c}^{s,HB}$, $\tilde{c}^{s,CR}$和 $\tilde{c}^{s,BR}$,并触发恢复.②恢复成功,有两种情形:(*a*)若$(FB\otimes TR)\parallel HB\parallel(CR\otimes BR)$所有子任务均成功执行,*TRP* 执行继续;(*b*)当$(FB\otimes TR)\parallel HB\parallel(CR\otimes BR)$子任务之一执行失败,安装$((FB'\otimes\varnothing)\parallel RH((FB\otimes TR)\parallel HB\parallel(CR\otimes BR),((FB'\otimes\varnothing)\parallel HB')\parallel(CR'\otimes BR'))$, $\tilde{c}^{s,par}$, *compnst _ trigger*),依次构造 $\tilde{c}^{s,FB'}$, $\tilde{c}^{s,HB'}$, $\tilde{c}^{s,CR'}$和 $\tilde{c}^{s,BR'}$上下文,并执行恢复.

(ⅲ)分支 $TDE\otimes TDU$ 执行失败,获取 $\tilde{c}^s$ 的外部激活条件,构造恢复上下文 $\tilde{c}^{s,TDE'}$(或 $\tilde{c}^{s,TDU'}$),结合选择聚合的恢复语义,执行恢复 $\zeta_{TDE\otimes TDU}|^{condition}$.

根据 TRP 可接受状态集的有效性,从协调的角度验证组合事务中任务之间的兼容性. TRP 执行时为了确保任务失败发生时任务间的兼容性和有效性,可以分别采用下列策略之一处理:(1)任务 TDE 执行失败后经过有限次重试后最终成功提交,TRP 继续正向执行;(2)当 *HB*、*CR* 或 *BR* 之一执行失败,且分别存在替代服务 HB_1、CR_1 和 BR_1,采用 *ARS* 执行恢复;(3)当任务 *OP* 执行失败是致命的(*fatal failure*),采用 BCPS 策略执行恢复;若 *OP* 执行失败是非致命的,则采用 FCPS 执行恢复后,TRP 继续正向执行.

6　结论和未来工作

面向服务环境中,组合事务的执行失败不可避免,基于组合事务恢复策略和执行日志,为事务失败恢复给出执行语义分析.我们组合事务系统的执行描述为格局变化过程,格局的变化必然引发事件,数据转移并修改组合事务的状态.这种基于扩展 Petri 网的组合事务的执行语义分析,结合模型检测技术可给出准确的执行语义:(1)分析不同任务的执行行为;(2)协调失败事务恢复;(3)给出组合事务准确的执行进展,避免歧义产生,确保组合事务的可靠执行.

未来的工作包括:(1)丰富事务执行日志;(2)引入事务流挖掘技术;(3)抽取更有效的数据流依赖;(4)优化事务恢复机制;(5)深入研究用户局部和全局约束,增强失败恢复的复杂决策,减少人工干预.

参考文献

[1] N B Lakhal, T Kobayashi, H Yokota. FENECIA: failure endurable nested-transaction based execution of composite Web services with incorporated state analysis [J]. International Journal on Very Large Data Bases, 2009, 18(1): 1 – 56.

[2] T Minh, N Le, J Cao. Flexible and semantics-based support for web services transaction protocols [A]. Proceedings of the 3rd International Conference on Advances in Grid and Pervasive Computing [C]. Kunming: Springer, 2008. 492 – 503.

[3] M Schäfer, P Dolog, W Nejdl. An environment for flexible advanced compensations of web service transactions [J]. ACM Transactions on the Web, 2008, 2(2): 1 – 36.

[4] S G Deng, Z Wu, et al. Modeling service compatibility with Π-calculus for choreography [A]. Proceedings of the 25th International Conference on Conceptual Modeling [C]. Tucson: Springer, 2006. 26 – 39.

[5] 殷昱煜,李莹,邓水光,尹建伟. Web 服务行为一致性与相容性判定[J]. 电子学报, 2009, 37(3): 433 – 438.
Yin Yu-yu, Li Ying, Deng Shui-guang, Deng Jian-wei. Determining on consistency and compatibility of web service behavior [J]. Acta Electronica Sinica, 2009, 37(3): 433 – 438. (in Chinese)

[6] Y Cheng, Z Wang, et al. Modeling and verifying composite semantic Web service based on colored Petri nets [A]. Proceedings of the 6th International Conference on Advanced Language

Processing and Web Information Technology [C]. Luoyang: IEEE, 2007. 510 – 514.

[7] 王勇,代桂平,侯亚荣,等.基于并发事务逻辑的 Web 服务编制验证[J].电子学报,2009,37(10):2228 – 2233.
Yong Wang, Gui-ping Dai, Ya-rong Hou, et al. Verification of web service orchestration based on concurrent transaction logic [J]. Acta Electronica Sinica, 2009, 37(10): 2228 – 2233. (in Chinese)

[8] K Wiesner, R Vaculín, et al. Recovery mechanisms for semantic web services [A]. Proceedings of the 8th IFIP WG 6.1 International Conference on Distributed Applications and Interoperable Systems [C]. Berlin: Springer, 2008. 100 – 105.

[9] J F He, H B Zhu, et al. A model for BPEL-like languages [J]. Frontiers of Computer Science in China, 2007, 1(1): 9 – 19.

[10] C Ferreira, I Lanese, et al. Advanced mechanisms for service combination and transactions [A]. Rigorous Software Engineering for Service-Oriented Systems Results of the SENSORIA Project on Software Engineering for Service-Oriented Computing [C]. Berlin: Springer, 2011. 302 – 325.

[11] H Y Sun, J Yang. Exploiting CoBTx-Net to verify the reliability of collaborative business transactions [A]. Proceedings of the 2nd IEEE Asia-Pacific Service Computing Conference [C]. Tsukuba: IEEE, 2007. 415 – 422.

[12] H Y Sun, J Yang. BTx-Net: A token based dynamic model for supporting consistent collaborative business transactions [A]. Proceedings of the IEEE International Conference on Services Computing [C]. Salt Lake City: IEEE, 2007. 415 – 422.

[13] Activebpel [OL]. http://www.activebpel.org/download, End-points corporation.

作者简介

梅晓勇 男.1974 年 3 月生,湖南常德人.博士,副教授,中国计算机学会 Petri 网专委会会员,1997 年在湖南师范大学获工学学士、2005 年和 2011 年先后在中山大学获工学硕士和博士学位.现为国家 Linux 培训与推广中心 Huas 分中心主任.研究兴趣包括面向服务计算、Petri 网技术和可信计算等.
E-mail: cdmxy@126.com

李师贤 男.1944 年 6 月生,江西于都人,教授,博士生导师.93 年确定为享受国务院政府特殊津贴专家,历任中山大学计算机科学系主任、信息科学与技术学院院长等职.主要从事软件工程和形式语义学的研究工作.

黄昌勤 男,1972 年 2 月生,湖南桃源人,博士,教授,博士生导师,IEEE 会员.1999 年和 2005 年分别在华东师范大学和浙江大学获工学硕士和博士学位.主要从事信息技术、服务计算和移动计算等方面的研究工作.

郑小林 男.1977 年 8 月生,浙江江山人,博士,副教授,现为浙江大学软件学院电子商务系执行系主任,IEEE 会员.主要从事电子商务、服务计算、移动计算等方面的研究工作.