ISSN 1673-9418 CODEN JKYTA8
Journal of Frontiers of Computer Science and Technology
1673-9418/2012/06(03)-0208-13
DOI: 10.3778/j.issn.1673-9418.2012.03.002

E-mail: fcst@vip.163.com
http://www.ceaj.org
Tel: +86-10-51616056

面向服务软件异常处理过程的可终止性验证*

蒋曹清[1,2+], 应 时[1], 文 静[1], 贾向阳[1], 管 华[1]

1. 武汉大学 软件工程国家重点实验室, 武汉 430072

2. 广西财经学院 信息与统计学院, 南宁 530003

Verification of Termination for Exception Handling Process in Service-Oriented Software*

JIANG Caoqing[1,2+], YING Shi[1], WEN Jing[1], JIA Xiangyang[1], GUAN Hua[1]

1. State Key Laboratory of Software Engineering, Wuhan University, Wuhan 430072, China

2. College of Information and Statistics, Guangxi University of Financial and Economics, Nanning 530003, China

+ Corresponding author: E-mail: jcqng@163.com

JIANG Caoqing, YING Shi, WEN Jing, et al. Verification of termination for exception handling process in service-oriented software. Journal of Frontiers of Computer Science and Technology, 2012, 6(3): 208–220.

Abstract: The dynamic and uncertainty of large-scale service-oriented software trend to cause high complexity of the exceptions logic which can lead to be extremely difficult to verify the termination of exception handling process. However, the termination of exception handling process is important foundation to ensure its correctness. If the exception handling process does not terminate, service-oriented software will not work normally. The current research rarely focuses on the verification method of termination for exception handling process in service-oriented software, thus exception handling can not be guaranteed to achieve the desired objectives. Therefore, this paper proposes a colored Petri net (CPN)-based verification method of termination for exception handling process in service-oriented software. Firstly, this method establishes the hierarchy CPN model for exception handling (HCPN4EH) including the CPN description of normal flow and exception handling logic. Then, the termination of exception handling process can be analyzed and verified according to the established model. Finally, an example demonstrates the feasibility and effectiveness of this method. The results of termination verification can provide the foundation for fur-

*The National Natural Science Foundation of China under Grant No. 61070012 (国家自然科学基金); the National Grand Basic Research 973 Program of China under Grant No. G2007CB310800 (国家重点基础研究发展规划(973)).

Received 2011-07, Accepted 2011-09.

ther analysis of correctness of the exception handling process.

Key words: exception handling; termination verification; verification method; service-oriented software

摘　要：大规模面向服务软件运行环境的动态性和不确定性使其异常处理逻辑复杂度高，导致异常处理过程的可终止性验证异常困难。而异常处理过程的可终止性是确保其正确性的重要基础，如果异常处理过程不能终止将导致面向服务软件无法正常运行。目前缺乏异常处理过程的可终止性验证方法，从而无法保证异常处理达到预期的目标。基于着色 Petri 网(colored Petri net, CPN)提出了一种面向服务软件异常处理过程的可终止性验证方法。该方法建立了包括正常流程和异常处理逻辑的异常层次 CPN 模型(hierarchy CPN model for exception handling, HCPN4EH)。基于此模型验证了异常处理过程的可终止性。通过一个实例说明了该方法的可行性和有效性。得到的可终止性验证结果可为进一步分析异常处理过程的正确性提供基础。

关键词：异常处理；可终止性验证；验证方法；面向服务软件

文献标识码：A　　**中图分类号：**TP311

1　引言

面向服务软件是通过将因特网上分布的服务资源组合起来，形成能满足用户需求的应用系统。由于面向服务软件的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，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，从而使其异常处理逻辑复杂度高，导致异常处理过程的可终止性验证异常困难。本文所指的异常处理过程的可终止性指异常处理逻辑中针对某个异常的处理，一定到达预期的成功处理或异常终止的状态。异常处理过程的可终止性是确保其正确性的重要基础，如果异常处理过程不能终止，将导致面向服务软件无法正常运行。而目前缺乏异常处理过程的可终止性验证方法，从而无法保证异常处理到达预期的目标。因此本文针对上述问题，研究面向服务软件的异常处理过程的可终止性验证方法。

业务流程执行语言(business process execution language, BPEL)是构建面向服务软件通用的标准流程编程语言。本文以 BPEL 作为载体，并在文献[1]给出的 BPEL 流程转化为着色 Petri 网(colored Petri net, CPN)模型的基础上，首先给出异常处理逻辑的形式化模型，然后在此基础上阐述面向服务软件异常处理过程的可终止性验证方法。由于面向服务软件和异常处理的复杂性，需要异常处理过程具有层次性，即异常处理过程通常由不同层次正常流程模块的相应异常处理模块组成，以完成对某个异常的处理。

目前 Petri 网已经成为主流的形式化验证工具[2–3]。由 Lohmann[4]和 Yang 等人[5]分别提出了将 BPEL 转换为工作流网和非层次着色 Petri 网模型的方法，然而它们均不能清晰表达系统的层次结构。丹麦奥尔胡斯大学 Jensen 等人[6–7]提出了一种着色 Petri 网。它是一种分层的高级 Petri 网，同时具有 Petri 网和编程语言的特点，有利于构建并发系统模型。通过利用层次着色 Petri 网合理建模面向服务软件，能够清晰地表达系统及异常处理过程的层次结构。着色 Petri 网提供自动化的仿真工具 CPN Tools，支持对系统行为的分析和验证。目前在验证异常处理过程的可终止性方面缺乏相关的研究工作，因此本文引入层次着色 Petri 网形式化方法，在 Yang 等人工作基础上，建立包含复杂的面向服务软件正常流程及其异常处理逻辑的异常处理层次 CPN 模型(hierarchy CPN model for exception handling, HCPN4EH)，并基于此模型对异常处理过程的可终止性进行分析和验证。该验证方法首先给出了异常处理过程的可终止性定义及其相应的判定方法；然后根据该判定方法，在 CPN Tools 工具的支持下，

采用状态空间分析方法实现异常处理过程可终止性验证; 最后通过一个实例说明本文方法的可行性和有效性。

本文组织结构如下：第 2 章给出异常处理逻辑的形式化模型; 第 3 章分析和验证异常处理过程的可终止性; 第 4 章给出一个实例, 说明本文方法的可行性和有效性; 第 5 章介绍相关的研究工作; 最后总结全文, 并指出未来工作方向。

2 基于 CPN 的异常处理逻辑的形式化模型

为了验证异常处理过程的可终止性, 需要基于 CPN 建立异常处理逻辑的形式化模型。此模型是由多个正常流程模块和异常处理逻辑模块按照给定的组织方式构成的层次化模型。正常处理模块可以由多个正常流程模块和异常处理逻辑模块构成, 且正常流程模块和异常处理逻辑模块又可以是嵌套定义的, 从而构成一套完整的 BPEL 流程的异常处理 CPN 模型 HCPN4EH。下面将按正常流程模块、异常处理逻辑模块、包含正常流程和异常处理逻辑的模型和包括异常处理的层次着色 Petri 网模型的顺序, 分别进行介绍。

2.1 正常流程模块的形式化模型

定义1 (正常流程模块的形式化模型) 正常流程模块是一个分层着色 Petri 网模块, 它是一个六元组 $CPN_{\mathrm{M}}=(CPN,T_{\mathrm{sub}},P_{\mathrm{port}},PT,CPN_{\mathrm{MP}},t_{\mathrm{MP}})$, 其中：

(1) $CPN=(P,T,F,\Sigma,C,M_0)$是一个描述模块 CPN_{M} N_{M} 的工作流网[5]。CPN 有两个特殊的库所 p_{s} 和 p_{f}。库所 p_{s} 是流程的开始库所, ${}^{\bullet}p_{\mathrm{s}}=\phi$, 表示正常处理流程的开始位置; 库所 p_{f} 是结束库所, $p_{\mathrm{f}}{}^{\bullet}=\phi$, 表示正常处理流程的结束位置。

(2) $T_{\mathrm{sub}}\subseteq T$ 是一个替代变迁集合。

(3) $P_{\mathrm{port}}\subseteq P$ 是一个端口库所集合。

(4) $PT:P_{\mathrm{port}}\rightarrow\{IN,OUT,I/O\}$是一个端口类型函数, 用来设置每个端口库所的类型。

(5) $CPN_{\mathrm{MP}}=(P_{\mathrm{mp}},T_{\mathrm{mp}},F_{\mathrm{mp}},\Sigma_{\mathrm{mp}},C_{\mathrm{mp}},M_{\mathrm{mp0}})$是 CPN_{M} 的父模块。

(6) t_{MP} 是 CPN_{M} 在父模块 CPN_{MP} 中所对应的替代变迁。

正常流程模块的形式化模型用来对面向服务软件中的各个模块进行建模。下面考虑面向服务软件中模块发生异常时, 如何对此模块相应的异常处理逻辑进行建模。

2.2 异常处理模块的形式化模型

本文把某个模块中针对某个异常进行处理的程序称为异常处理模块。下面定义异常处理模块的形式化模型。

定义2 (异常处理模块的形式化模型) 异常处理模块是一个分层着色 Petri 网模块, 它是一个六元组 $CPN_{\mathrm{EM}}=(CPN_{\mathrm{E}},T_{\mathrm{esub}},P_{\mathrm{eport}},PT_{\mathrm{e}},CPN_{\mathrm{EMP}},t_{\mathrm{EMP}})$, 其中：

(1) $CPN_{\mathrm{E}}=(P_{\mathrm{e}},T_{\mathrm{e}},F_{\mathrm{e}},\Sigma_{\mathrm{e}},C_{\mathrm{e}},M_{\mathrm{e0}})$是一个描述异常处理模块 CPN_{EM} 的工作流网。CPN_{E} 有两个特殊的库所 p_{es} 和 p_{ef}。库所 p_{es} 是异常处理模块的开始库所, ${}^{\bullet}p_{\mathrm{es}}=\phi$, 表示异常处理模块的开始位置; 库所 p_{ef} 是结束库所, $p_{\mathrm{ef}}{}^{\bullet}=\phi$, 表示异常处理模块的结束位置。

(2) $T_{\mathrm{esub}}\subseteq T_{\mathrm{e}}$ 是一个替代变迁的集合。

(3) $P_{\mathrm{eport}}\subseteq P_{\mathrm{e}}$ 是一个端口库所集合。

(4) $PT:P_{\mathrm{eport}}\rightarrow\{IN,OUT,I/O\}$是一个端口类型函数, 用来设置每个端口库所的类型。

(5) $CPN_{\mathrm{EMP}}=(P_{\mathrm{emp}},T_{\mathrm{emp}},F_{\mathrm{emp}},\Sigma_{\mathrm{emp}},C_{\mathrm{emp}},M_{\mathrm{emp0}})$是 CPN_{EM} 的父模块。

(6) t_{EMP} 是 CPN_{EM} 在父模块 CPN_{EMP} 中所对应的替代变迁。

当正常流程出现异常时, 此模型用来对产生异常的模块的相应异常处理模块进行建模。其中, 异常处理模块 CPN_{EM} 可以正常终止于结束库所。如果异常处理过程中又重新抛出异常, 则在异常发生变迁到结束库所之间建立一条有向弧, 并用异常 Token 表示异常终止。下面将通过正常流程模块和异常处理模块的模块融合来建立形式化模型。

2.3 异常处理模块与正常流程模块融合的形式化模型

为了描述异常处理模块与正常流程模块的融合, 首先需要说明异常处理过程, 然后根据异常处理的不同情况, 在融合时建立不同的模型。

2.3.1 异常处理过程

步骤 1 当正常流程某个活动发生异常时，则在此活动发生异常的相应变迁处，添加一个外向弧和一个描述异常的库所 P_e(表示异常抛出)。

步骤 2 如果发生异常的模块有相应的异常处理模块，则融合异常处理模块的开始库所 p_{es} 与异常库所P_e(Token 颜色相同)，表示捕获异常并进行异常处理。

步骤 2.1 如果异常处理成功，则异常处理正常终止于异常处理模块的结束库所 P_f，融合 P_f 与将要返回的库所(它们的Token 颜色相同)，返回到正常流程相应位置继续执行。

步骤 2.2 否则异常处理异常终止于异常处理模块的结束库所 P_f。首先把异常处理模块的结束库所 P_f 设为端口库所，把发生异常活动所在模块的父模块中异常处理模块的相应变迁处添加一个外向弧和一个描述异常的库所 P_e(表示异常抛出)，并把此库所 P_e 作为槽库所(与 P_f 的 Token 颜色相同)，表示抛出此异常到父模块。父模块的异常捕获与异常处理过程与步骤 2 所述相同。

步骤 3 如果发生异常的模块没有相应的异常处理模块，则此正常模块异常终止于结束库所，异常抛出到父模块。异常处理过程与步骤 2.2 所述类似。

步骤 4 对于下层向上抛出异常，如果本层有相应的异常处理模块，则按步骤 2 所述过程进行异常处理。否则继续向上抛出异常，过程与步骤 3 所述相同，直到顶层模块。如果顶层模块也没有相应的异常处理模块，则 BPEL 流程异常终止于此流程模块的结束库所。

2.3.2 模块融合的形式化模型

定义3 (异常处理成功时模块融合的形式化模型)给定正常流程模块 $CPN_M=(CPN,T_{sub},P_{port},PT,CPN_{MP},t_{MP})$，其中$CPN=(P,T,F,\Sigma,C,M_0)$；异常处理模块 $CPN_{EM}=(CPN_E,T_{esub},P_{eport},PT_e,CPN_{EMP},t_{EMP})$，其中 $CPN_E=(P_e,T_e,F_e,\Sigma_e,C_e,M_{e0})$。要求$CPN_{MP}=CPN_{EMP}$，在 CPN_M 中添加异常生成库所 p_e 及发生异常的变迁到 p_e 的弧 f_e，p_e 与异常模块的开始库所 p_{es} 是融合库所；如果异常处理成功，则异常模块的终止库所 p_{ef} 与正常模块 CPN_M 中异常返回的库所 p_r 为融合库所。即 $C(p_e)=C_e(p_{es})$, $M_0(p_e)=M_{e0}(p_{es})$; $C_e(p_{ef})=C(p_r)$, $M_{e0}(p_{ef})=M_0(p_r)$。那么 $N_C=(P_c,T_c,F_c,\Sigma_c,C_c,M_{c0},T_{csub},P_{cport},PT_c,CPN_{MP},t_{MP},CPN_{EMP},t_{EMP})$是 CPN_M 和 CPN_E 的融合，当且仅当满足下列要求：

(1) $P=P\cup\{p_e\}$, $P_c=P\cup P_e$;

(2) $T_c=T\cup T_e$, $T\cap T_e=\phi$;

(3) $F=F\cup\{f_e\}$, $F_c=F\cup F_e$;

(4) $\Sigma_c=\Sigma\cup\Sigma_e$;

(5) $\forall p\in P_c$，如果$p\in P$, $C_c(p)=C(p)$，否则 $C_c(p)=C_e(p)$;

(6) $\forall p\in P_c$，如果 $p\in P$，$M_{c0}(p)=M_0(p)$，否则 $M_{c0}(p)=M_{e0}(p)$;

(7) $T_{csub}=T_{sub}\cup T_{esub}$;

(8) $P_{cport}=P_{port}\cup P_{eport}$;

(9) $PT_c=PT\cup PT_e$。

还需要在父模块 CPN_{MP} 中添加槽库所 p_{sps}(设为 p_e、p_{es} 的槽库所)、p_{spf}(设为 p_{ef}、p_r 的槽库所)，然后添加 t_{MP} 到 p_{sps} 的弧 f_{MPsps} 与 p_{sps} 到 t_{EMP} 的弧 f_{spsEMP}，添加 t_{EMP} 到 p_{spf} 的弧 f_{EMPspf} 与 p_{spf} 到 t_{MP} 的弧 f_{spfMP}，并设置好端口 p_e、p_{es}、p_{ef}、p_r 的类型。即：

(1) $P_{emp}=P_{emp}\cup\{p_{sps},p_{spf}\}$

$F_{emp}=F_{emp}\cup\{f_{MPsps},f_{spsEMP},f_{EMPspf},f_{spfMP}\}$

(2) $PT(p_e)=OUT$, $PT(p_{es})=IN$

$PT(p_{ef})=OUT$, $PT(p_r)=IN$

定义4 (异常处理失败时模块融合的形式化模型)给定正常流程模块$CPN_M=(CPN,T_{sub},P_{port},PT,CPN_{MP},t_{MP})$，其中 $CPN=(P,T,F,\Sigma,C,M_0)$；异常处理模块$CPN_{EM}=(CPN_E,T_{esub},P_{eport},PT_e,CPN_{EMP},t_{EMP})$，其中 $CPN_E=(P_e,T_e,F_e,\Sigma_e,C_e,M_{e0})$。要求$CPN_{MP}=CPN_{EMP}$，在 CPN_M 中添加异常生成库所 p_e 及发生异常的变迁到 p_e 的弧 f_e，p_e 与异常模块的开始库所 p_{es} 是融合库所；如果异常处理不成功，则异常处理模块的结束库所 p_{ef} 与 CPN_{EMP} 中模块 CPN_{EM} 所对应的变迁 t_{EMP} 处添加的库所 p_{ep} 为融合库所。即 $C(p_e)=C_e(p_{es})$, $M_0(p_e)=$

$M_{e0}(p_{es})$; $C_e(p_{ef})=C_{emp}(p_{ep})$, $M_{e0}(p_{ef})=M_{emp0}(p_{ep})$。那么 $NC=(P_c,T_c,F_c,\Sigma_c,C_c,M_{c0},T_{csub},P_{cport},PT_c,CPN_{MP},t_{MP},CPN_{EMP},t_{EMP})$是 CPN_M和 CPN_E的融合，当且仅当满足下列要求：

(1) $P=P\cup\{p_e\}$, $P_c=P\cup P_e$;

(2) $T_c=T\cup T_e$, $T\cap T_e=\phi$;

(3) $F=F\cup\{f_e\}$, $F_c=F\cup F_e$;

(4) $\Sigma_c=\Sigma\cup\Sigma_e$;

(5) $\forall p\in P_c$, 如果$p\in P$, $C_c(p)=C(p)$, 否则 $C_c(p)=C_e(p)$;

(6) $\forall p\in P_c$, 如果 $p\in P$, $M_{c0}(p)=M_0(p)$, 否则 $M_{c0}(p)=M_{e0}(p)$;

(7) $T_{csub}=T_{sub}\cup T_{esub}$;

(8) $P_{cport}=P_{port}\cup P_{eport}$;

(9) $PT_c=PT\cup PT_e$。

且需要在父模块 CPN_{MP}中添加槽库所 p_{sps}(设为 p_e、p_{es}的槽库所)、p_{sef}(设为 p_{ef}的槽库所)，然后添加 t_{MP}到 p_{sps}的弧f_{MPsps}与 p_{sps}到 t_{EMP}的弧f_{spsEMP}，添加 t_{EMP}到 p_{sef}的弧 f_{EMPsef}，并设置好端口 p_e、p_{es}、p_{ef}的类型。即：

(1) $P_{emp}=P_{emp}\cup\{p_{sps},p_{sef}\}$

$F_{emp}=F_{emp}\cup\{f_{MPsps},f_{spsEMP},f_{EMPsef}\}$

(2) $PT(p_e)=OUT$, $PT(p_{es})=IN$, $PT(p_{ef})=OUT$

2.4 异常处理的层次 CPN 模型

下面将基于 2.1~2.3 节建立的正常流程模块与异常处理模块的融合模型，构建异常处理的层次 CPN 模型，然后给出各模块间的层次结构。

定义 5 (异常处理的层次 CPN 模型) 异常处理的层次着色 Petri 网是一个四元组$CPN_{EH}=(S,SM,PS,FS)$，其中：

(1) S 是模块的有限集，每个模块都是一个着色 Petri 网模块 $s=(CPN^s,P^s_{sub},T^s_{port},PT^s)$。对任意 $s_1,s_2\in S$，且 $s_1\neq s_2$，满足$(P^{s_1}\cup T^{s_1})\cap(P^{s_2}\cup T^{s_2})=\phi$。

(2) $SM:T_{sub}\to S$ 是子模块函数，为每个替换变迁设置子模块，要求模块层次是非循环的。

(3) PS 是一个端口-槽关联函数，此函数为每个替代变迁 t 指定一个端口-槽关联关系 $PS(t)\subseteq P_{sock}(t)\times P^{SM(t)}_{port}$。对任意的$(p,p')\in PS(t)$, $t\in T_{sub}$，要求 $PT(p)=PT(p')$, $C(p)=C(p')$，并且 $M_0(p)=M_0(p')$。

(4) $FS\subseteq 2^P$ 是非空融合集组成的集合，对任意的$p,p'\in fs$和任意$fs\in FS$，有$C(p)=C(p')$，并且 $M_0(p)=M_0(p')$。

定义6 (模块间的层次结构) 异常处理的层次着色 Petri 网 $CPN_{EH}=(S,SM,PS,FS)$的模块层次是一个有向图 $MH=(N_{MH},F_{MH})$。$N_{MH}=S$ 是描述模块的节点集合，$F_{MH}=\{(s_1,t,s_2)\in N_{MH}\times T_{sub}\times N_{MH}|t\in T^{s_1}_{sub}\wedge s_2=SM(t)\}$是弧的集合。$MH$的根称为主模块，所有主模块的集合记为 S_{PM}。

在上述各种模型的基础上可建立包含正常流程及其异常处理逻辑的层次模型 HCPN4EH。下文将在该模型的基础上介绍异常处理可终止性验证方法。

3 异常处理过程的可终止性验证

基于已建立的形式化模型 HCPN4EH，首先给出异常处理过程的可终止性定义，并分析得到可终止性判定规则；接着给出可终止性判定算法，并使用仿真工具 CPN Tools，采用状态空间分析方法对上述可终止性判定规则进行验证，从而实现对异常处理过程的可终止性验证。

3.1 异常处理过程的可终止性判定规则

异常处理过程是由一些异常处理模块组成的，因此判定异常处理过程的可终止性转化为判定这些异常处理模块的可终止性，然后在此基础上考虑整个异常处理过程的可终止性。根据程序的可终止性验证方法[8]，本文把异常处理模块的可终止性分为可终止和不可终止，异常处理过程的可终止性分为可终止和不可终止。下面首先对它们进行概念界定。

定义7 (异常处理模块的可终止性) 在异常处理模块 CPN_{EM}中，如果从 CPN_{EM}的开始库所 p_{es}存在异常 Token，则模块经过一系列变迁，Token 总会到达结束库所 p_{ef}。称满足此条件的异常处理模块 CPN_{EM}是可终止的，否则是不可终止的。即：

$\exists p_{es} \in CPN_{EM}$, $p_{ef} \in CPN_{EM}$, 如果p_{es}所含的 Token 数量$|M(p_{es})|>0$, 经过有限次触发异常处理变迁后, 必有p_{ef}所含的 Token 的数量$|M(p_{ef})|>0$, 则此CPN_{EM}是可终止的。

在定义 7 的基础上, 设某个异常 e 的异常处理需经过 n 个异常处理模块$CPN_{EM}^{1}, CPN_{EM}^{2}, \ldots, CPN_{EM}^{n}$依次处理, 则异常 e 的异常处理过程的起始库所为 $p_{es}^{1} \in CPN_{EM}^{1}$, 结束库所为 $p_{ef}^{n} \in CPN_{EM}^{n}$。异常处理过程的可终止性定义为：

定义 8 (异常处理过程的可终止性) 在异常 e 的异常处理过程 EP 中, 如果起始库所 p_{es}^{1} 存在异常 Token, 则该过程经过一系列变迁, Token 总会到达结束库所 p_{ef}^{n}。称满足此条件的异常处理过程 EP 是可终止的, 否则是不可终止的。即：

$\exists p_{es}^{1} \in CPN_{EM}^{1}, p_{ef}^{n} \in CPN_{EM}^{n}$, 如果$p_{es}^{1}$所含的Token 数量$|M(p_{es}^{1})|>0$, 经过有限次触发异常处理变迁后, 必有 p_{ef}^{n} 所含的 Token 的数量$|M(p_{ef}^{n})|>0$, 则此 EP 是可终止的。

根据上述异常处理可终止性相关概念, 下面将给出异常处理过程的可终止性判定规则。

判定规则 1 (异常处理模块的可终止判定) 如果异常处理模块 CPN_{EM} 的状态空间中满足$|M(p_{ef})|>0$ 的标识 M_{ef} 既是家标识又是死标识, 则异常处理模块 CPN_{EM} 可终止。即：

(1) 设$|M(p_{es})|>0$ 的状态是初始标识 M_0, M_{ef} 是家标识当且仅当如果 $\forall M \in R(M_0)$, 那么一定有 $M_{ef} \in R(M)$。其中 $R(M_0)$表示从初始标识 M_0 可达的标识, $R(M)$表示从 M 可达的标识。

(2) M_{ef} 是死标识当且仅当如果 $\forall t \in T$, 那么有 $\neg(M_{ef} \xrightarrow{t})$。其中 t 表示变迁, T 表示变迁集, $M_{ef} \xrightarrow{t}$ 表示在 M_{ef} 下 t 是使能的。

判定规则2 (异常处理模块的不可终止判定) 如果异常处理模块 CPN_{EM} 的状态空间中 M_{ef} 不是家标识, 或者 M_{ef} 不是死标识, 则异常处理模块 CPN_{EM} 不可终止。即：

(1) 设$|M(p_{es})|>0$ 的状态是初始标识 M_0, M_{ef} 不是家标识当且仅当 $\exists M \in R(M_0)$, 有 $M_{ef} \notin R(M)$。

(2) M_{ef} 不是死标识当且仅当如果 $\exists t \in T$, 那么有 $M_{ef} \xrightarrow{t}$。

判定规则 3 (异常处理过程的可终止判定) 如果异常处理过程 EP 中所有的异常处理模块都是可终止的, 则异常处理过程 EP 可终止。即：

设某个异常 e 的异常处理过程 EP 需要经过 n 个 $CPN_{EM}^{1}, CPN_{EM}^{2}, \ldots, CPN_{EM}^{n}$ 异常处理模块依次处理, 如果对于任意 CPN_{EM}^{i}, 若有 p_{es}^{i} 所含的Token 数量$|M(p_{es}^{i})|>0$, 必有 p_{ef}^{i} 所含的 Token 数量$|M(p_{ef}^{i})|>0$, 则 EP 是可终止的。

判定规则 4 (异常处理过程的不可终止判定) 如果异常处理过程 EP 中存在不可终止的异常处理模块, 则 EP 不可终止。即：

设某个异常 e 的异常处理过程 EP 需要经过 n 个 $CPN_{EM}^{1}, CPN_{EM}^{2}, \ldots, CPN_{EM}^{n}$ 异常处理模块依次处理, 如果存在 CPN_{EM}^{i}, 若有 p_{es}^{i} 所含的 Token 数量$|M(p_{es}^{i})|>0$, 不能使 p_{ef}^{i} 所含的 Token 数量$|M(p_{ef}^{i})|>0$, 则 EP 是不可终止的。

下面将根据异常处理可终止性定义及判定规则, 采用状态空间分析方法来验证异常处理可终止性。

3.2 异常处理过程的可终止性验证过程

本文将使用 CPN Tools 工具的状态空间分析方法来验证异常处理可终止性。为此, 基于系统的层次 CPN 模型 HCPN4EH 的图形描述, 首先计算需要验证的异常处理过程的各个异常处理模块的状态空间; 然后根据异常处理模块可终止性判定规则得到的判定算法进行验证; 最后根据异常处理过程可终止性判定规则进行验证。

根据系统的层次 CPN 模型 HCPN4EH 的定义, 容易通过工具 CPN Tools 得到此模型的图形描述。由于异常处理的复杂性和层次性, 异常处理过程一般由一个或多个异常处理模块组成, 如果能够对异常处理过程中每一个异常处理模块进行验证, 则可根据判定规则 3、4 实现异常处理过程的可终止性验证。因此首先需要探讨异常处理模块的可终止验

证方法。

根据异常处理可终止性判定规则，可得到如算法 1 所示的异常处理模块的可终止性判定算法。

算法 1 异常处理模块的可终止性验证算法

输入：异常处理模块计算得到的状态空间及初始标识 M_0。

输出：异常处理模块的可终止性判定结果。

1．$NODES \leftarrow \{M_0\}$

2．$UNPROCESSED \leftarrow \{M_0\}$

3．$ARCS \leftarrow \phi$

4．while $UNPROCESSED \neq \phi$ do

5． Select a marking M in $UNPROCESSED$

6． $UNPROCESSED \leftarrow UNPROCESSED - \{M\}$

7． if M has no arc $(M,(t,b),M')$ such that $M \xrightarrow{(t,b)} M'$ then exit end if //要求没有处理的点一定不是死标识，即死标识只能是 M_{ef}

8． for all arc $(M,(t,b),M')$ such that $M \xrightarrow{(t,b)} M'$ do

9． if M' is not M_{ef} then

10． if $M' \notin NODES$ then

11． $NODES \leftarrow NODES \cup \{M'\}$

12． $UNPROCESSED \leftarrow UNPROCESSED \cup \{M'\}$

13． end if

14． end if

15． if M' is M_{ef} and $\exists M''$ such that $M' \xrightarrow{t} M''$ then exit end if //要求 M_{ef} 是死标识

16． end for

17．end while

算法 1，的时间复杂性为 $n \times e$，其中 n 为状态空间图的结点数，e 为状态空间图的边数。

通过算法 1 可验证每一条从满足 $|M(p_{es})|>0$ 的初始标识 M_0 开始的所有可能路径的终点都是满足 $|M(p_{ef})|>0$ 的标识 M_{ef}，并验证标识 M_{ef} 没有后继标识，即标识 M_{ef} 既是家标识也是死标识，从而实现异常处理模块的可终止性验证。

假设某个异常 e 的异常处理需要经过 n 个 $CPN_{EM}^1, CPN_{EM}^2, \ldots, CPN_{EM}^n$ 异常处理模块依次处理，图 1 给出了异常处理过程可终止性验证过程。

异常处理过程的可终止性验证过程如下：

(1) 利用 CPN Tools 工具构建第 i 个异常处理模块的状态空间。

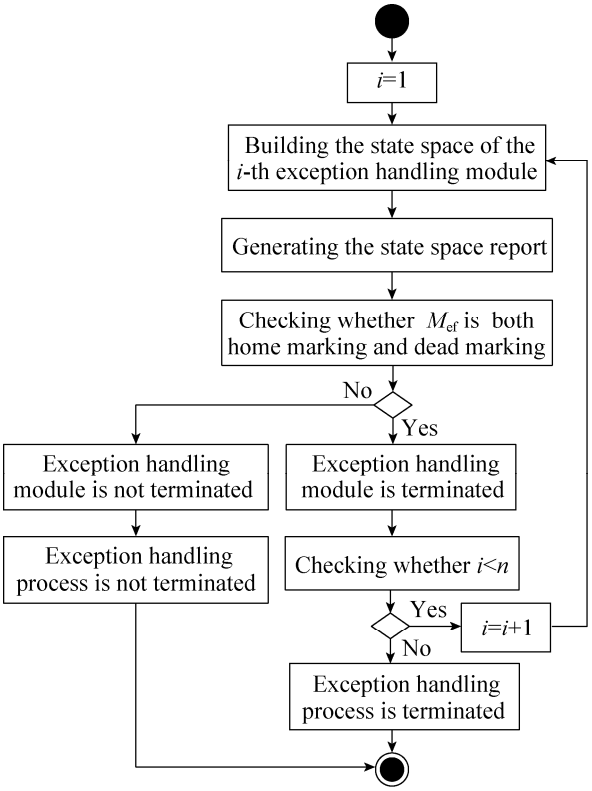


Fig.1 Termination verification process

图 1 可终止性验证过程

(2) 基于已计算的状态空间，并使用算法 1 生成状态空间报告。

(3) 分析状态空间报告，检查标识 M_{ef} 是否既是家标识又是死标识。

(4) 如果是，则说明异常处理模块是可终止的；否则说明异常处理模块是不可终止的，根据判定规则 4 可判定异常处理过程是不可终止的，验证结束。

(5) 检查是否满足条件 $i<n$。

(6) 如果 $i<n$，回到步骤(1)验证第 $i+1$ 个异常处理模块。

(7) 如果 $i \geq n$，则根据判定规则 3 判定此异常处理过程是可终止的，验证结束。

4 案例研究

本文以电子商务领域的在线食品商店为案例展开研究[9]。在线食品商店是一个销售和配送食品的公司，这个公司由一个在线商店和一些位于不同地区的仓库组成。每一个仓库都与本地供货商有联系，

负责存储不易变质的食品，并向附近的顾客进行实际交付。客户通过下订单、支付账单以及接收食品与食品商店进行交互。如果订购的食品不易保存或者缺货，则食品公司必须联系多个供货商。在线食品商店的 Shop 和 Supplier 伙伴的流程描述如图 2 所示。

在线食品商店系统可能发生的异常按异常发生位置分为顾客异常、商店异常、供应商异常和仓库异常等四类。如供应商异常有：(1)由于预订代码不被供应商认可(可能在供应商的 CancelOrder 活动上或者供应商的 ConfirmOrder 活动上，见图 2)而产生的异常 WrongResCodeException；(2)由于买家商店一直没有告知供应商是否取消订单或者进行下一步活动(Shop CancelOrder 活动之后，见图2)而产生的异常 TimeOutException。

下面假设供应商在接收从商店发送的取消预订请求时，在供应商数据库中没有此预订代码。对于此异常，供应商采取如下异常处理步骤：(1)通知商店重新发送取消预订请求 UnReservationRequest；(2)重新接收取消预订请求 UnReservationRequest；(3)进行取消预订操作。如果步骤(2)接收的取消预订请求仍然不被认可，则说明异常处理不成功，需要把此异常抛出到上层模块由商店进行处理，其异常处理步骤为：(1)要求顾客重新输入；(2)更新数据库；

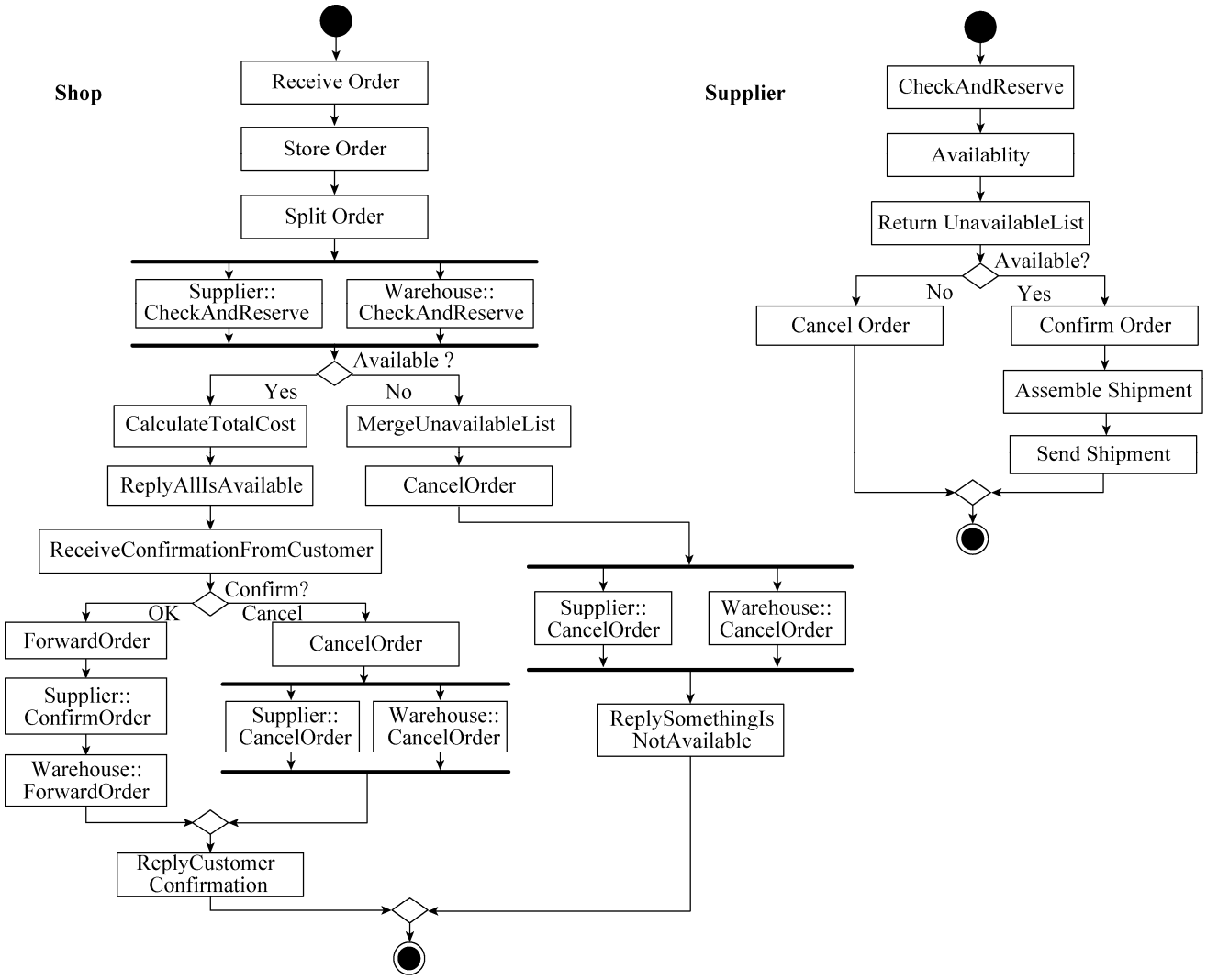


Fig.2 Flow diagram of Shop and Supplier partners in online food shop example

图 2 在线食品商店案例的 Shop 与 Supplier 伙伴流程图

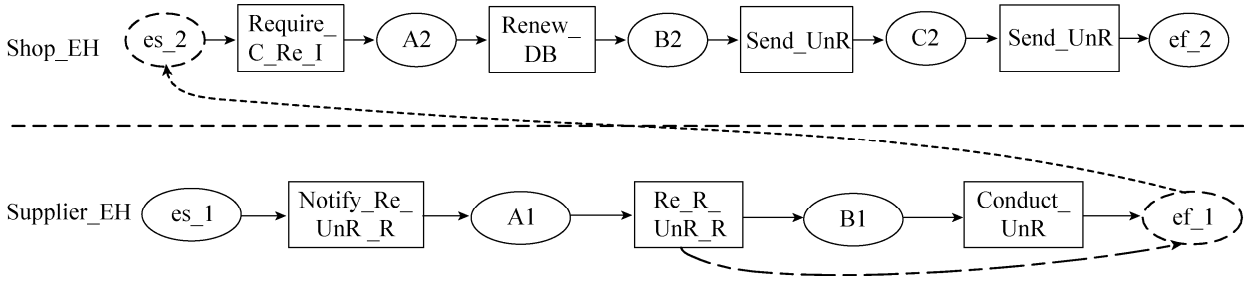


Fig.3 Exception handling process diagram when to send cancellation request

图 3 发送取消预订请求时的异常处理过程示意图

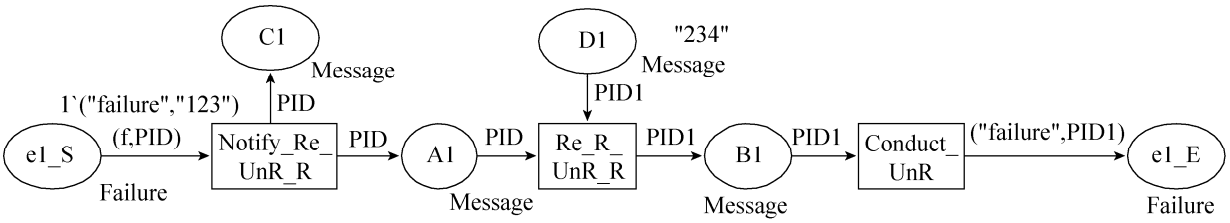


Fig.4 CPN description for exception handling module Supplier_EH

图 4 异常处理模块 Supplier_EH 的 CPN 描述

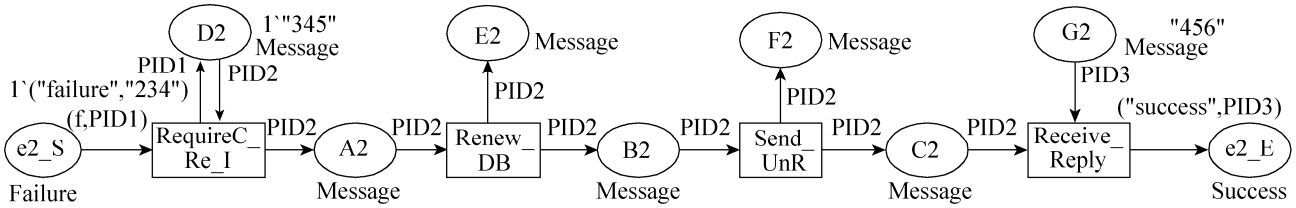


Fig.5 CPN description for exception handling module Shop_EH

图 5 异常处理模块 Shop_EH 的 CPN 描述

(3)向供应商重新发送取消预订请求 UnReservation-Request; (4)接收供应商回复取消成功消息。此异常处理过程依次由 Supplier_EH 和 Shop_EH 异常处理模块组成(如图 3 所示)。

图 3 中 ef_1 和 es_2 是融合库所，也就是说 ef_1 和 es_2 看成同一库所。图 3 说明，首先由 Supplier 伙伴中的异常处理模块 Supplier_EH 进行异常处理。由于异常没有成功处理，通过 ef_1 库所与 es_2 库所的融合，把此异常抛出到 Shop 伙伴，并由 Shop_EH 异常处理模块来处理。

要验证此异常处理过程的可终止性，需要依次分别验证 Supplier_EH、Shop_EH 异常处理模块的可终止性。图 4、图 5 分别是 Supplier_EH、Shop_EH 异常处理模块的 CPN 模型。然后根据此 CPN 模型计算状态空间，并分别得到分析报告的部分截图(如图 6、图 7 所示)。从分析报告可以得到两个模块的终止标识 M_{ef}=[16]、M_{ef}=[20]既是家标识又是死标识。根据判定规则 1 判定两个模块都是可终止的，进而根据判定规则 3 判定此异常处理过程也是可终止的。

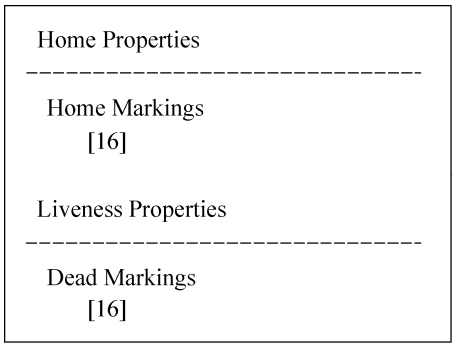


Fig.6 State space report for Supplier_EH: home properties and liveness properties

图 6 Supplier_EH 的状态空间报告的部分截图

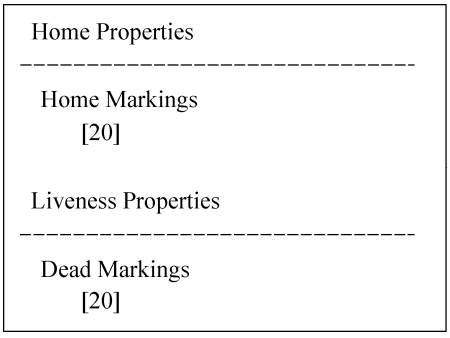


Fig.7 State space report for Shop_EH: home properties and liveness properties

图 7 Shop_EH 的状态空间报告的部分截图

5 相关工作

近年来，随着 Web 服务技术的飞速发展，用 BPEL 等流程语言编写面向服务软件逐渐成为一种新的开发范式。但由于面向服务软件所处环境复杂多变，导致软件运行时容易产生异常。如何对异常处理建模、如何确保异常处理的可终止性等问题逐渐引起了国内外研究者的关注。

为了进一步分析和验证异常处理过程的各种性质，如正确性、可终止性等，需要对异常处理过程进行建模。国内外主要相关研究有：Brambilla 等人[10]给出了工作流驱动的 Web 应用中常见异常的分类方法，并针对各种类型的异常设计了合适的处理方法，能够在较高抽象层次上建模 Web 应用流程及其异常处理过程。文献[11]提出了一种层次型协同监控模型(hierarchical collaborative monitor model of migrating workflow, HCM3)，通过监控者之间的协调机制在多个层次上诊断与处理异常状态，以保持全局状态一致。该模型能够实现监控的并发性和异常处理的层次性，提高了工作流执行的可靠性。文献[12]提出了一种基于 Petri 网，对异常处理过程中的元素(即活动、异常事件、异常处理策略以及处理措施)进行形式化描述的方法，特别是对不同处理策略下的处理过程进行了描述。同时结合消息机制，分析了一个异常事件出现时，如何处理一个活动实例产生的中间数据，并对事务处理和异常处理的补偿策略进行了比较。文献[13]通过扩展 Petri 网中的基本概念，在将工作流内在的逻辑关系分解为控制流与数据流的基础上，提出了一个面向异常处理的工作流系统复合建模方法。对应工作流的模型执行，定义了两类异常处理，用矩阵形式分别表示或标识控制流、数据流和异常事件，经过矩阵分析与变换，对异常事件的影响区域进行范围界定。Friedrich 等人[14]针对基于服务的流程中的异常处理和修复问题，将与服务调用相关的故障类型划分为返回错误信息的故障和由于偶发性错误或数据暂时性失配所导致的、不返回任何信息的故障。以上研究建立了各种异常处理模型，也对一些性质进行了分析，但与本文不同的是，他们没有对系统中的异常处理进行建模，且没有考虑异常处理的可终止验证问题。

异常处理逻辑的分析与验证方面代表性的工作有：Hamadi等人[15]通过验证自适应恢复网(self-adapting recovery net, SARN)的一致性约束(如可达性、活性和有界性)，保证了工作流异常处理行为的正确性。Brito 等人[16]提出了一种基于 B 方法和通信顺序进程(communicating sequential processes, CSP)的异常传播形式化分析与验证方法，并使用该方法对其提出的一种容错软件体系结构进行了形式化验证。文献[17]提出了一种基于 B 方法和 Alloy 规约语言的异常流分析与验证方法，并使用该方法对分布式系统中的协同异常处理机制进行了形式化分析与验证。Pereira 等人[18]基于协调原子动作理论，提出了一种带有异常处理协调机制的体系结构模型，用于指导并发容错系统的形式化规约。该模型提供一组 CSP 进程和预定义通道，规约和协调组件的异常处理，并利用 FDR(failure divergence refinement)验证与协调原子动作和并发异常处理相关的安全属性。文献[19]使用 Petri 网判断服务行为和服务的一致性。文献[20]提出了一种包括异常处理结构的函数间控制流图的构建方法。该方法把引发异常的隐式控制流显式地表示出来，异常传播路径在控制流图中也一目了然。在此基础上，提出了基于异常传播分析的 C++程序的数据流分析方法，并给出了相应的算法，有助于实现数据流分析的自动处理。Wang 等人[21]提出了一种基于组件的软件体系结构的建模方法、约束定义以及一致性验证方

法。该方法使用约束组件的计算树逻辑(component constraint computational tree logic, C2-CTL)描述软件约束和设计组成, 然后将 C2-CTL 映射到 Petri 网模型, 并通过 Petri 网模型验证系统级约束和组件约束的一致性。文献[22]提出了一种事务性 Web 服务的容错组合框架(framework for fault-tolerant composition of transactional Web services, FACTS)。该框架提供了一种融合异常处理和事务技术的混合型容错机制(exception handling + transaction, EXTRA), 并给出了异常处理逻辑正确性的验证算法。文献[23]提出了一种基于 Petri 网的服务组合故障诊断与处理方法。该方法通过分析服务组合的故障需求, 给出服务组合故障处理的框架, 并采用 Petri 网来解决服务组合的错误发现及其处理问题。文献[24]提出了扩展着色 Petri 网的模型检测方法, 并将扩展后的着色 Petri 网模型检测方法及工具应用在 Web 服务组合的验证问题中, 能够验证 Web 服务组合是否存在逻辑错误。此外程序的可终止性验证问题已经有较多的研究成果[8,25]。上述这些研究中, 对于面向服务软件异常处理描述及异常处理过程的可终止性方面的研究非常缺乏。

6 结束语

本文针对异常处理过程的可终止验证问题, 提出了一种基于层次着色 Petri 网的形式化验证方法。首先, 建立包括正常流程和异常处理过程的层次 CPN 模型 HCPN4EH, 此模型不仅能够建模面向服务软件的正常流程和异常处理逻辑, 而且能够建模它们的层次结构。然后, 给出异常处理模块、异常处理过程的可终止性定义及其可终止性判定规则, 并通过使用形式化验证工具 CPN Tools 对异常处理过程的可终止性进行验证。最后, 结合在线食品商店实例, 说明了异常处理过程可终止验证方法的可行性和有效性。验证得到的可终止性结果可为进一步分析异常处理过程的正确性提供基础。下一步的工作主要包括两个方面：

(1) 引入仿真分析方法进一步完善异常处理可终止性验证方法。

(2) 研究异常处理过程正确性的验证方法。

References:

[1] Tan Wei, Fan Yushun, Zhou Mengchu. A Petri net-based method for compatibility analysis and composition of Web services in business process execution language[J]. IEEE Transactions on Automation Science and Engineering, 2009, 6(1): 94–106.

[2] van der Aalst W M P. The application of Petri nets to workflow management[J]. Journal of Circuits, Systems and Computers, 1998, 8(1): 21–66.

[3] Adam N, Alturi V, Huang W K. Modeling and analyzing of workflows using Petri nets[J]. Journal of Intelligent Information Systems, 1998, 10(2): 131–158.

[4] Lohmann N. A feature-complete Petri net semantics for WS-BPEL 2.0[C]//Dumas M, Heckel R. Proceedings of the 4th International Conference on Web Services and Formal Methods (WS-FM ’07). Berlin, Heidelberg: Springer-Verlag, 2007: 77–91.

[5] Yang Yanping, Tan Qingping, Yu Jinshan, et al. Transformation BPEL to CP-nets for verifying Web services composition[C]//Abraham A, Han Sang Yong, Du Hung-Chang, et al. Proceedings of the International Conference on Next Generation Web Services Practices (NWESP ’05). Washington, DC, USA: IEEE Computer Society, 2005: 137–142.

[6] Jensen K. An introduction to the theoretical aspects of coloured Petri nets[C]//de Bakker J W, de Roever W P, Rozenberg G, et al. LNCS 803: A Decade of Concurrency, Reflections and Perspectives, REX School/Symposium. London, UK: Springer-Verlag, 1994: 230–272.

[7] Jensen K, Kristensen L M, Wells L. Colored Petri nets and modeling and validation of concurrent systems[J]. International Journal on Software Tools for Technology Transfer, 2007, 9(3/4): 213–254.

[8] Colon M A, Sipma H B. Practical methods for proving program termination[C]//Brinksma E, Larsen K G. LNCS 2404: Proceedings of the 14th International Conference on Computer Aided Verification (CAV ’02). London, UK:

Springer-Verlag, 2002: 442–454.

[9] Console L, Fugini M. WS-DIAMOND: an approach to Web service-diagnosability, monitoring, and diagnosis[C]// Cunningham P, Cunningham M. Proceedings of the International e-Challenges Conference. Amsterdam: IOS Press, 2007: 105–112.

[10] Brambilla M, Comai S, Tziviskou C. Exception management within Web applications implementing business processes[J]. Advanced Topics in Exception Handling Techniques (LNCS), 2006, 4119(4): 101–120.

[11] Lu Zhaoxia, Zeng Guangzhou. A cooperative monitoring model of migrating workflow[J]. Journal of Computer Research and Development, 2009, 46(3): 398–406.

[12] Sun Ruizhi, Shi Meilin. Formal presentation of exception handling in a workflow system[J]. Journal of Computer Research and Development, 2003, 40(3): 393–397.

[13] Dou Wanchun, Xi Xiaopeng, Xu Liefei, et al. Exception handling oriented workflow modeling and its performance[J]. Chinese Journal of Computers, 2003, 26(9): 1094–1103.

[14] Friedrich G, Fugini M, Mussi E, et al. Exception handling for repair in service-based processes[J]. IEEE Transactions on Software Engineering, 2010, 36(2): 198–215.

[15] 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.

[16] Brito P, Lemos R, Rubira C, et al. Architecting fault tolerance with exception handling: verification and validation[J]. Journal of Computer Science and Technology, 2009, 24(2): 212–237.

[17] Castor Filho F, Romanovsky A, Rubira C M F. Improving reliability of cooperative concurrent systems with exception flow analysis[J]. Journal of Systems and Software, 2009, 82(5): 874–890.

[18] Pereira D P, de Melo A C V. Formalization of an architectural model for exception handling coordination based on CA action concepts[J]. Science of Computer Programming, 2010, 75(5): 333–349.

[19] Hu Hao, Yin Qin, Lv Jian. Service behavior and quality consistency in virtualized computing environment[J]. Journal of Software, 2007, 18(8): 1943–1957.

[20] Jiang Shujuan, Xu Baowen, Shi Liang. An approach of data-flow analysis based on exception propagation analysis[J]. Journal of Software, 2007, 18(1): 74–84.

[21] Wang Jiacun, Zhou Xianzhong, Ding Junhua. Software architectural modeling and verification: a Petri net and temporal logic approach[J]. Transactions of the Institute of Measurement and Control, 2011, 33(1): 1–14.

[22] Liu An, Li Qing, Huang Liusheng, et al. FACTS: a framework for fault-tolerant composition of transactional Web services[J]. IEEE Transactions on Services Computing, 2010, 3(1): 46–59.

[23] Fan Guisheng, Yu Huiqun, Chen Liqiong, et al. Fault diagnosis and handling for service composition based on Petri nets[J]. Journal of Software, 2010, 21(2): 231–247.

[24] Men Peng, Duan Zhenhua. Extension of model checking tool of colored Petri nets and its applications in Web service composition[J]. Journal of Computer Research and Development, 2009, 46(8): 1294–1303.

[25] Bradley A R, Manna Z, Sipma H B. Termination of polynomial programs[C]//Cousot R. LNCS 3385: Proceedings of the 6th International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI 2005). Heidelberg: Springer, 2005: 113–129.

附中文参考文献：

[11] 卢朝霞, 曾广周. 面向迁移工作流可靠执行的协同监控模型[J]. 计算机研究与发展, 2009, 46(3): 398–406.

[12] 孙瑞志, 史美林. 工作流异常处理的形式描述[J]. 计算机研究与发展, 2003, 40(3): 393–397.

[13] 窦万春, 席晓鹏, 许列飞, 等. 面向意外处理的工作流系统建模与执行[J]. 计算机学报, 2003, 26(9): 1094–1103.

[19] 胡昊, 殷琴, 吕建. 虚拟计算环境中服务行为与质量的一致性[J]. 软件学报, 2007, 18(8): 1943–1957.

[20] 姜淑娟, 徐宝文, 史亮. 一种基于异常传播分析的数据流分析方法[J]. 软件学报, 2007, 18(1): 74–84.

[23] 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.

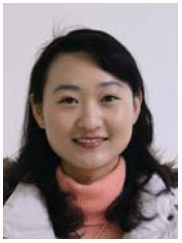
[24] 门鹏, 段振华. 着色 Petri 网模型检测工具的扩展及其在 Web 服务组合中的应用[J]. 计算机研究与发展, 2009, 46(8): 1294–1303.

JIANG Caoqing was born in 1973. He is a Ph.D. candidate at Wuhan University. His research interests include formal method, program analysis and software architecture.

蒋曹清(1973—), 男, 湖南永州人, 武汉大学博士研究生, 主要研究领域为形式化方法, 程序分析, 软件体系结构。

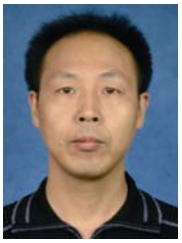
YING Shi was born in 1965. He received his Ph.D. degree in computer software and theory from Wuhan University in 1999. Now he is a professor and Ph.D. supervisor at Wuhan University. His research interests include service-oriented and aspect-oriented software development, semantic Web technologies, software architecture and patterns, software reusability and interoperability, etc.

应时(1965—), 男, 湖北武汉人, 1999 年于武汉大学计算机软件与理论专业获得博士学位, 现为武汉大学教授、博士生导师, 主要研究领域为面向服务和面向对象的软件开发, 语义 Web 技术, 软件体系结构和模式, 软件的可重用性与互操作性等。



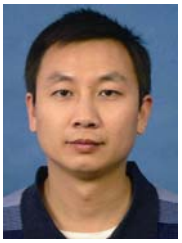
WEN Jing was born in 1982. She received her Ph.D. degree in computer software and theory from Wuhan University in 2011. Her research interests include software architecture, aspect-oriented software development and human computer interaction.

文静(1982—), 女, 湖北武汉人, 2011 年于武汉大学计算机软件与理论专业获得博士学位, 现为武汉大学博士后, 主要研究领域为软件体系结构, 面向方面软件开发, 人机交互。



JIA Xiangyang was born in 1972. He received his Ph.D. degree in computer software and theory from Wuhan University in 2008. Now he is a lecturer at Wuhan University. His research interests include service-oriented software development, service-oriented middle component and semantic technologies, etc.

贾向阳(1972—), 男, 湖北襄阳人, 2008 年于武汉大学计算机软件与理论专业获得博士学位, 现为武汉大学讲师, 主要研究领域为面向服务的软件开发, 面向服务的中间件, 语义技术等。



GUAN Hua was born in 1978. He is a Ph.D. candidate at Wuhan University. His research interests include software architecture and formal method.

管华(1978—), 男, 湖北武汉人, 武汉大学博士研究生, 主要研究领域为软件体系结构, 形式化方法。