DOI：10.7652/xjtuxb201304020

面向服务软件中异常处理的形式化建模方法

蒋曹清[1,2]，应时[1]，文静[1]，贾向阳[1]，王一兵[3]
（1.武汉大学软件工程国家重点实验室，430072，武汉；2.广西财经学院信息与统计学院，530003，南宁；
3.第二炮兵指挥学院三系，430012，武汉）

摘要：针对面向服务软件中异常处理难以建模的问题，基于层次着色Petri网提出了一种面向服务软件中异常处理的形式化建模方法。该方法从异常的抛出、捕获、处理、返回或传播等方面入手进行了异常处理成分建模，给出了各成分的层次化的形式语义模型，来清晰地表达异常处理的过程。结合实例从有效性和可靠性角度对建模方法进行了验证，结果表明，所提方法能够充分利用着色Petri网的层次和数据类型的建模能力，为面向服务软件的数据流和控制流、大规模面向服务软件的层次化模型建模，为异常处理性质的分析与验证提供支持。

关键词：面向服务软件；异常处理；形式化建模；建模方法
中图分类号：TP311 **文献标志码**：A **文章编号**：0253-987X(2013)04-0118-07

A Modeling Approach for Exception Handling in Service-Oriented Software

JIANG Caoqing[1,2], YING Shi[1], WEN Jing[1], JIA Xiangyang[1], WANG Yibing[3]
(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; 3. No. 3 Department, The Second Artillery Command College, Wuhan 430012, China)

Abstract: Aiming at the difficulty of modeling exception in service-oriented software, a formal modeling approach is proposed to solve the problem based on colored Petri nets (CPN). With the proposed approach, exception handling constructs in service-oriented software are modeled to cover exception throwing, capturing, handling, return and propagation. Then, the formal semantics are specified for the hierarchical models of exception handling constructs to describe the exception handling process clearly. A case study is given to verify the effectiveness and soundness of the proposed modeling approach. The results show that the approach can fully take advantage of modeling capabilities of colored Petri net for hierarchy and data type to effectively model data flow and control flow in a service-oriented software, to establish a hierarchical model of exception handling in large-scale service-oriented software, and to provide support for the analysis and verification of properties relating to exception handling.

Keywords: service-oriented software; exception handling; formal modeling; modeling approach

面向服务软件是指通过将因特网上分布的服务资源组合起来形成能够满足用户需求的应用系统。由于面向服务软件的松耦合性、服务资源的自治性、运行环境的动态性和不确定性，使得软件在运行过

收稿日期：2012-07-07。 作者简介：蒋曹清(1973—)，男，博士生，副教授；应时(通信作者)，男，教授，博士生导师。 基金项目：国家"九七三"重点基础研究发展规划资助项目(2007CB310800)；国家自然科学基础重点资助项目(91118003，61272113，61272108)；国家自然科学基金资助项目(61070012，61170022)。
网络出版时间：2012-12-23 网络出版地址：http://www.cnki.net/kcms/detail/61.1069.T.20131223.1637.009.html

程中会面临许多异常情况，需要更复杂的异常处理机制，从而导致异常处理的形式化建模很困难。

业务流程执行语言(BPEL)是构建面向服务软件的通用标准流程编程语言[1]，相关研究涉及到BPEL流程的多种形式化建模方法[2-4]，如：Lohmann和Ouyang等人给出了BPEL完整的Petri网语义[5-8]，但其异常处理描述很复杂，或者仅能刻画异常处理逻辑控制流，从而导致验证非常困难；Yang等人建立了将BPEL转换为工作流网或非层次着色Petri网(CPN)的方法[9-10]，但缺乏异常处理的层次化建模能力。由于BPEL中引入了一些特殊语言元素，所以其语义与一般语言元素有所不同，在处理方式上有时不能将异常抛出到上一层作用域(scope)。本文针对上述问题，提出了一种面向服务软件中异常处理的建模方法。

1　建模基础

BPEL语言结构包括基本活动和结构化活动，建模BPEL时首先建立基本活动的CPN模型，在此基础上依照结构化活动的成分及语义建立相应的CPN模型。

1.1　基本活动的建模

基本活动是BPEL流程中的原子活动，建模时涉及到控制流和数据流。下面给出各种模型在共同颜色类型集下的定义。

定义1　颜色类型集 Σ 定义为

```
colset  STRING=string;
colset  CONTROL=unit;
colset  PARAMETER=record v_name:
        STRING * v_value: STRING;
colset  EXCEPTION=record e_name:
        STRING * e_i: STRING;
colset  MSG=union emsg: EXCEPTION+
        msg: PARAMETER
```

CPN中颜色类型也称为颜色集。颜色集中：CONTROL的实例用来描述流程的控制权；PARAMETER为活动的输入输出参数类型，其中 v_{name} 为输入输出变量名，v_{value} 为输入输出变量值，均用字符串表示；EXCEPTION用来刻画活动抛出的异常信息，其中 e_{name} 为异常名，e_i 为异常上下文信息，均用字符串表示；MSG为活动中输入输出消息的类型，根据实际应用可定义为异常消息或返回结果。本文中未作特别说明的颜色类型集均为 Σ。

根据基本活动的语义，基本活动在执行前需要输入变量或输入消息，在执行后产生输出变量或输出消息。

定义2　基本活动的CPN模型 $\Psi_{BA}=(P^{BA},T^{BA},F^{BA},\Sigma^{BA},V^{BA},C^{BA},G^{BA},E^{BA},M_0^{BA})$，是一个着色Petri网，如图1所示。

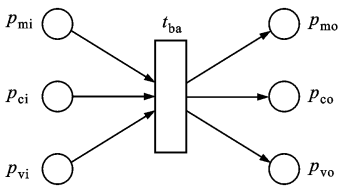

图1　基本活动的CPN模型

建模时：每一个BPEL基本活动功能属性需建成变迁 t_{BA}；活动触发条件建成 t_{BA} 的入库所 p_{ci}，颜色类型为CONTROL；活动发生所需的变量参数建成 t_{BA} 的入库所 p_{vi}，颜色类型为PARAMETER；活动的输入消息建成变迁 t_{BA} 的入库所 p_{mi}，颜色类型为MSG；活动结束建成变迁 t_{BA} 的出库所 p_{co}，颜色类型为CONTROL；活动发生后生成数值参数建成 t_{BA} 的出库所 p_{vo}，颜色类型为PARAMETER；活动输出消息建成 t_{BA} 的出库所 p_{mo}，颜色类型为MSG。

1.2　结构化活动的建模

结构化活动的CPN建模是根据BPEL流程结构化活动的语义及其包含的基本活动情况，将基本活动连接起来，如图2所示。

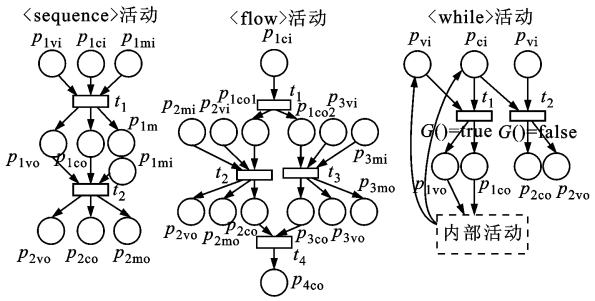

图2　结构化活动的CPN模型

scope是BPEL流程中最重要的结构化活动，可采用自底向上的方式对scope逐层建模。如果scope中抛出异常，则建模时要在scope中建立异常的抛出、处理、捕获和返回或传播等语义模型。

2　异常处理的建模方法

2.1　异常抛出的建模

异常抛出建模如下。

(1)用一个异常库所 p_e 为活动抛出的异常建模，其颜色类型为EXCEPTION，库所中的token为

抛出的异常类型，并用变量 e 表示，库所中不同的 token 表示不同的异常类型，从描述该活动的变迁 t 处引出一条指向 p_e 的弧，表示异常被抛出。

(2)用一个控制库所 p_{eco} 为活动发生异常后获得控制流的控制权建模，其颜色类型为 CONTROL，从描述该活动的变迁 t 处引出一条指向 p_{eco} 的弧$\langle t, p_{eco}\rangle$，表示异常发生后控制流的控制权将偏移至异常处理逻辑。

(3)针对从下层 scope 到上层 scope 抛出的异常，异常抛出变迁为下层 scope 在其上层 scope 中的替代变迁，建模方法与上述(1)、(2)类似。

〈invoke〉活动异常抛出的 CPN 模型如图 3 所示。该异常来自被〈invoke〉活动调用的 Web 服务的非正常运行。本文研究被调用 Web 服务描述语言(WSDL)文档中显式声明异常抛出情形下的〈Invoke〉活动语义模型。为了简化描述，本文假定一个基本活动只抛出一种异常。

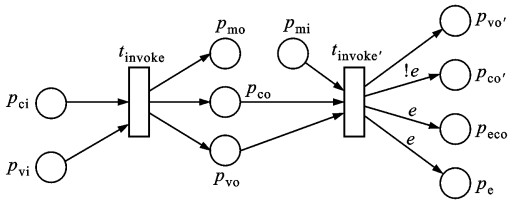

图 3 〈invoke〉活动异常抛出的 CPN 模型

定义 3 〈invoke〉活动异常抛出的语义模型是一个 9 元组，即 $\Psi_{invoke}=(P,T,F,\Sigma,V,C,G,E,M_0)$，其中：$P=\{p_{ci},p_{vi},p_{mo},p_{mi},p_{vo},p_{co},p_{vo'},p_{co'},p_e,p_{eco}\}$；$T=\{t_{invoke},t_{invoke'}\}$，$t_{invoke}$ 与 $t_{invoke'}$ 表示活动依次发生；$F=\{\langle p_{ci},t_{invoke}\rangle,\langle p_{vi},t_{invoke}\rangle,\langle t_{invoke},p_{mo}\rangle,\langle t_{invoke},p_{co}\rangle,\langle t_{invoke},p_{vo}\rangle,\langle p_{vo},t_{invoke'}\rangle,\langle p_{co},t_{invoke'}\rangle,\langle p_{mi},t_{invoke'}\rangle,\langle t_{invoke'},p_{co'}\rangle,\langle t_{invoke'},p_{vo'}\rangle,\langle t_{invoke'},p_e\rangle,\langle t_{invoke'},p_{eco}\rangle\}$；$V=\{v:\text{PARAMETER};m:\text{MSG};e:\text{EXCEPTION};c:\text{CONTROL}\}$；$C=\{C(p_{ci})=\text{CONTROL},C(p_{co})=\text{CONTROL},C(p_{vi})=\text{PARAMETER},C(p_{vo})=\text{PARAMETER},C(p_{mo})=\text{MSG},C(p_{mi})=\text{MSG},C(p_{co'})=\text{CONTROL},C(p_{eco})=\text{CONTROL},C(p_e)=\text{EXCEPTION}\}$；$G(t)=\text{true},t\in T$；$E=\{E(\langle p_{ci},t_{invoke}\rangle)=c,E(\langle p_{vi},t_{invoke}\rangle)=v,E(\langle t_{invoke},p_{mo}\rangle)=m,E(\langle t_{invoke},p_{co}\rangle)=c,E(\langle t_{invoke},p_{vo}\rangle)=v,E(\langle p_{co},t_{invoke'}\rangle)=c,E(\langle p_{vo},t_{invoke'}\rangle)=v,E(\langle p_{mi},t_{invoke'}\rangle)=m,E(\langle t_{invoke'},p_{co'}\rangle)=c,E(\langle t_{invoke'},p_{vo'}\rangle)=v,E(\langle t_{invoke'},p_{eco}\rangle)=c,E(\langle t_{invoke'},p_e\rangle)=e\}$；$M_0(p_{ci})=1'(),M_0(p_{vi})=1'v,M_0(p_{mi})=1'm,\forall p\in P:p\neq p_{ci},p\neq p_{vi},p\neq p_{mi}\rightarrow M_0(p)=\text{NULL}$。

〈throw〉活动是指，业务流程显式地抛出一个异常，并提供异常名，选择性地提供关于异常的更多的信息数据。〈throw〉活动的 CPN 模型如图 4 所示。

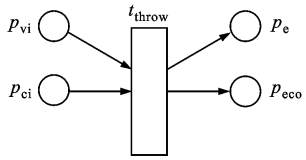

图 4 〈throw〉活动的 CPN 模型

scope 中隐式定义的异常抛出的建模方法：根据 scope 中定义的、异常处理器能处理的异常，首先在 scope 中确定可抛出异常的活动，然后进行描述，具体描述方法与上述显式异常抛出的描述方法相同。

2.2 异常处理逻辑片段的建模

异常处理逻辑片段是指，BPEL 中错误处理器(faulthandlers)的〈catch〉活动、〈catchAll〉活动定义的异常处理逻辑。为了简化叙述，本文将异常处理逻辑片段模型称为异常处理逻辑模块，其对应于 faulthandlers 所在 scope 的上层 scope 模型中的相应替代变迁。

〈catch〉和〈catchAll〉活动中包含了进行异常处理的各种 BPEL 活动，各种活动的建模方法与第 1 节建模方法的不同之处在于，〈catch〉、〈catchAll〉活动中第一个活动变迁均添加了一个入库所 p_{es}，称之为异常处理逻辑模块的开始库所，前者库所中的 token 描述能处理异常类型及相关信息，后者库所中的 token 值暂为 NULL。

〈rethrow〉活动是指，在异常处理程序中重新抛出异常，包括原异常的异常名、异常相关信息，仅用于 faulthandlers 的〈catch〉和〈catchAll〉活动。〈rethrow〉必须忽略异常的修改信息，即在〈catch〉和〈catchAll〉活动中，如果先修改逻辑异常信息，再执行〈rethrow〉，则原来的异常信息被抛出，而不是抛出修改后的异常信息。图 5 是〈rethrow〉活动的 CPN 模型。

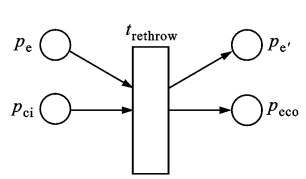

图 5 〈rethrow〉活动的 CPN 模型

2.3　异常捕获的建模

异常捕获是指，为面向服务软件中抛出的异常唯一地确定其对应的异常处理逻辑片段。

描述异常捕获需满足：p_e 的颜色集与 p_{es} 的颜色集相同，均为 EXCEPTION 类型；p_{eco} 的颜色集与 p_{ecs}（描述异常处理逻辑片段控制流的开始，即为异常处理逻辑模型中第一个变迁的 p_{ci}）的颜色集相同，均为 CONTROL；在父模块 CPN 模型中先为 p_e、p_{es} 添加槽库所 p_{sps}，为 p_{eco}、p_{ecs} 添加槽库所 p_{epc} 和替代变迁 t_{EMP}（匹配此异常的异常处理模块），再建立抛出异常的活动所在 scope 的正常模块 t_{MP} 到 p_{sps} 的弧〈t_{MP}，p_{sps}〉，p_{sps} 到 t_{EMP} 的弧〈p_{sps}，t_{EMP}〉，t_{MP} 到 p_{epc} 的弧〈t_{MP}，p_{epc}〉，p_{epc} 到 t_{EMP} 的弧〈p_{epc}，t_{EMP}〉；设置 p_e、p_{es}、p_{eco}、p_{epc} 的端口类型。

定义 4　异常捕获的语义可描述为：

（1）$C(p_e)=C_e(p_{es})=$EXCEPTION，$M_0(p_e)=M_{e0}(p_{es})$；

（2）$C(p_{eco})=C_e(p_{ecs})=$CONTROL，$M_0(p_{eco})=M_{e0}(p_{ecs})$；

（3）$P_{emp}=P_{emp}\cup\{p_{sps},p_{epc}\}$，$F_{emp}=F_{emp}\cup\{\langle t_{MP},p_{sps}\rangle,\langle p_{sps},t_{EMP}\rangle,\langle t_{MP},p_{epc}\rangle,\langle p_{epc},t_{EMP}\rangle\}$，$T_{esubp}=T_{esubp}\cup\{t_{EMP}\}$；

（4）$T_P(p_e)=$OUT，$T_P(p_{es})=$IN，$T_P(p_{eco})=$OUT，$T_P(p_{ecs})=$IN。

异常捕获描述中 $T_P(\cdot)$ 表示一个端口库所指定的端口类型，P_{emp} 为父模块 CPN 模型的库所集，F_{emp} 为父模块 CPN 模型的弧集，T_{esubp} 为父模块 CPN 模型的替代变迁集。

2.4　异常返回及异常传播的建模

如果异常处理能够完成异常处理逻辑模块的最后一个活动，则 t_{EMP} 将返回到此异常抛出 scope 的下一个活动，建立一条弧〈t_{EMP}，p_{return}〉，t_{EMP} 将与异常抛出 scope 的下一个活动的入库所 p_{return} 连接。描述异常成功返回需满足：能够对完成异常处理模块的最后一个活动进行描述的库所 p_{ecf} 和 p_{return} 的类型均为 CONTROL；p_{return} 为 p_{ecf} 的槽库所；将 t_{EMP} 添加到 p_{return} 的弧〈t_{EMP}，p_{return}〉，设置 p_{ecf} 的类型。

定义 5　一个抛出的异常被成功处理后返回的语义可描述为：

（1）$C_e(p_{ecf})=C(p_{return})=$CONTROL，$M_{e0}(p_{ecf})=M_0(p_{return})$；

（2）$F_{emp}=F_{emp}\cup\{\langle t_{EMP},p_{return}\rangle\}$；

（3）$T_P(p_{ecf})=$OUT。

异常处理时，如果执行到〈throw〉、〈rethrow〉活动，或该执行发生异常，则称此情形为异常处理弱终止。该异常将被抛到上层 scope，称此情形为异常传播，对应的建模方法如下。

（1）在 t_{EMP} 中，用一个 p_{en} 为新抛出的异常建模，颜色类型为 EXCEPTION，并用库所中的 token 为抛出的异常类型建模，从描述该活动的变迁 t_{en} 处引出一条指向 p_{en} 的弧〈t_{en}，p_{en}〉，以表示异常抛出。

（2）用一个 p_{eno} 为在活动发生异常后获得的流程控制权建模，其颜色类型为 CONTROL，并从 t_{en} 处引出一条指向 p_{eno} 的弧〈t_{en}，p_{eno}〉，表示异常发生后控制流的控制权将偏移至新的异常处理逻辑。

（3）在 t_{EMP} 的所在父模块 CPN 模型中建立一个 p_{en} 的槽库所 p_{ep} 和一条从 t_{EMP} 到 p_{ep} 的弧〈t_{EMP}，p_{ep}〉，表示异常 p_{en} 已从父模块 CPN 模型中抛出。

（4）在 t_{EMP} 的所在父模块 CPN 模型中建立一个 p_{eno} 的槽库所 p_{enop} 和一条从 t_{EMP} 到 p_{enop} 的弧〈t_{EMP}，p_{enop}〉，表示异常发生后控制流的控制权将从父模块 CPN 模型中传出。

（5）设置 p_{en}、p_{eno} 的端口类型为 OUT。

定义 6　一个抛出的异常经非成功处理后传播的语义可描述为：

（1）$C_e(p_{en})=C_{emp}(p_{ep})=$EXCEPTION，$C_e(p_{eno})=C_{emp}(p_{enop})=$CONTROL；

（2）$P_{em}=P_{em}\cup\{p_{en},p_{eno}\}$，$F_{em}=F_{em}\cup\{\langle t_{en},p_{en}\rangle,\langle t_{en},p_{eno}\rangle\}$；

（3）$P_{emp}=P_{emp}\cup\{p_{ep},p_{enop}\}$，$F_{emp}=F_{emp}\cup\{\langle t_{EMP},p_{ep}\rangle,\langle t_{EMP},p_{enop}\rangle\}$；

（4）$T_P(p_{en})=$OUT，$T_P(p_{eno})=$OUT。

采用以上异常处理建模方法，可以获得面向服务软件中异常处理的形式化模型。

3　验证实验

本文以汽车装配流水线管理系统为研究实例，对建模方法的有效性进行了验证，描述如下。

（1）根据本文建模方法，用 CPN Tools 对实例系统 17 个流程中 14 个带有异常处理逻辑的流程进行了建模，用计算状态空间工具计算得到了模型的状态空间报告，包括模型的有界性（boundedness properties）、家态性（home properties）、活性（liveness properties）、公平性（fairness properties）等基本性质。工厂资源配置流程模型的状态空间部分报告为

Home Properties

Home Markings

[4838]

Liveness Properties

Dead Markings

[4868]

Dead Transition Instances

None

Live Transition Instances

None

(2)检查状态空间报告中的基本性质,该性质应满足以下情况:①不存在上界和下界值均为 0 的库所,因为库所上界为 0 说明此库所从未有 token 值,即模型中该库所是多余的;②不存在 2 个家标识(home marking),或者家标识中结束库所的 token 值为 0,因为家标识是指从初始标识可达的任意标识都能到达的标识,其只能是流程的唯一结束状态;③死标识(dead marking)与家标识相同,因为死标识是指没有后继的标识,意味着流程执行终止(死标识)所对应的状态为结束状态(家标识);④不存在死变迁,因为死变迁意味着模型中存在从不使能的变迁,即模型中该变迁是多余的;⑤不存在活变迁,因为活变迁意味着模型存在永远使能的变迁,即模型中该变迁使得别的变迁不使能;⑥不存在公平性标识,因为公平性标识意味着流程存在无限执行序列,说明流程中存在某活动无限次的重复执行。

(3)针对流程中的异常先对模型进行初始配置,再用 CPN Tools 实施仿真分析,确认执行情况与原流程预计执行的一致性。

(4)对于非常复杂的异常处理过程,基于模型检测的思想进行验证[11]。

下面将以整车装配工位数据采集流程中的缺件异常为例对异常处理的可终止性进行验证。该流程中缺件异常处理过程依次由 MissingMaterials_EH 和 NoInBOM_EH 模块组成。图 6、图 7 分别为 MissingMaterials_EH、NoInBOM_EH 模块的 CPN 模型。

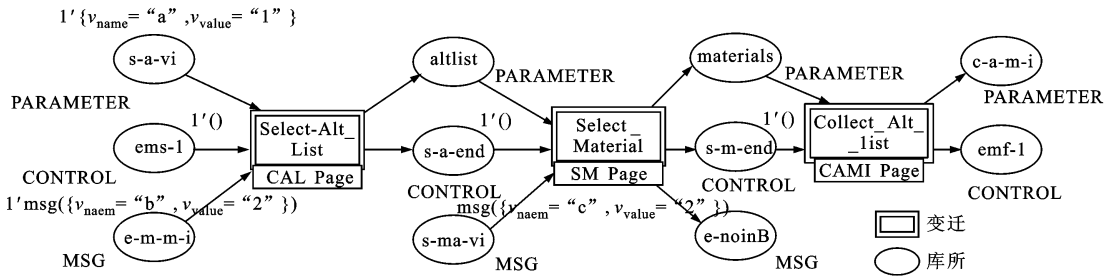

图 6 Missing Materials_EH 模块的 CPN 模型

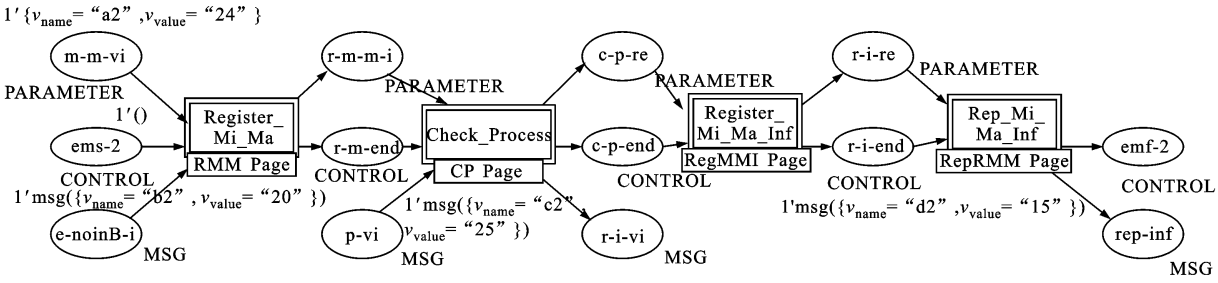

图 7 NoInBOM_EH 模块的 CPN 模型

为了验证异常处理的可终止性,用 CPN Tools 中的标准查询函数对 MissingMaterials_EH、NoInBOM_EH 模块的可终止性进行了检测,检测过程描述如下。

(1)计算异常处理模块 MissingMaterials_EH 的状态空间。

(2)用函数 fun ListDeadMarkings (): Node list=PredAllNodes Terminal 列出状态空间中的所有死标识,结果为[46],表示状态空间节点 46 是唯一死标识,它对应于异常抛出库所 p_{eno} 不为空的状态,说明在该状态下,异常处理流程运行终止。

(3)用函数 fun HomeMarking (n:Node): bool = SccTerminal (NodeToScc (n)) andalso HomeMarkingExists()检测得到上述检测过程(2)中列出

的节点 46 是家标识。由检测过程(2)、(3)可知，节点 46 处为弱终止。

(4)用类似方法检测 NoInBOM_EH 模块的状态空间，结果是：状态空间节点 72 是唯一的既是家标识，又是死标识的节点，该节点既不是弱终止节点，也不是强终止节点，由此可判定 NoInBOM_EH 模块是正常终止，从而验证“缺件”的异常处理是可终止的。

为了证实本文建模方法的适用性，以电子商务领域的在线食品商店为例展开了研究[12-13]。该商店包含 FoodShop、WarehouseServer 等共 26 个流程，其中带有异常处理逻辑的 21 个流程用本文方法建模，并从类似汽车装配流水线管理系统实例的验证过程发现，FoodShop 流程的状态空间报告中存在 3 个死变迁，说明该变迁的使能条件永远不能满足，所以该模型存在错误，经核实该错误为异常处理逻辑错误。

为了验证上述结果的可靠性，采用下面 2 种方法进行：① 依照传统的 Lohmann 方法[5]，用 BPEL2oWFN 工具建立了汽车装配流水线管理系统的工作流网模型，再用 Fiona 工具验证该系统的通信行为，由结果可以看出，Lohmann 方法通信行为与本文方法一致；②依照传统的 Ouyang 方法[8]，用 BPEL2PNML 工具建立了电子商务领域的在线食品商店的 Petri 网模型，再用 Woflan 工具分析了系统的合理性，由结果可以看出，Ouyang 方法与本文方法在流程的合理性方面是相同的。

图 8 为 Lohmann、Ouyang 及本文方法，在构建工厂资源配置流程 FRCP、整车装配工位数据采集流程 VSDAP、食品商店流程 FSP 和仓库服务流程 WSP 时，模型节点总数比较。

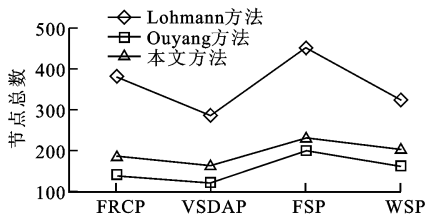

图 8　3 种方法在流程异常处理中模型节点总数的比较

由图 8 表明，构建异常处理模型时，用 Lohmann 方法所建库所和变迁总数较大，用 Ouyang 方法所建库所和变迁总数较小，用本文建模方法所建规模中等。本文建模方法可以充分利用着色 Petri 网的层次和数据类型的建模能力，为面向服务软件的数据流和控制流进行有效建模，且适用于大规模面向服务软件的层次化模型。

图 9 为 Lohman、Ouyang 及本文方法在计算状态空间的时间比较。

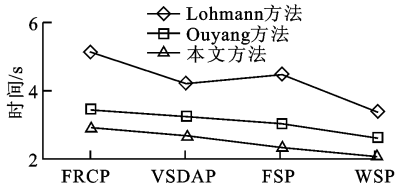

图 9　3 种方法在计算状态空间的时间比较

由图 9 表明，在计算状态空间的时间开销方面，从大到小依次为 Lohmann、Ouyang 和本文方法。由于本文方法基于了模块化建模思想，即在计算状态空间时可避免重复计算，所以时间开销小。目前，本文方法还未完全实现自动化，因而在建模的性能方面还不能给出全面的比较结果。

4　结　论

从异常的抛出、捕获、处理、返回或传播等方面对异常处理进行了建模，并给出了形式语义模型，其可为形式化分析与验证异常处理性质提供支持。通过典型实例对方法的有效性和可靠性进行了验证，结果表明：本文方法能够为面向服务软件中异常处理的形式化模型建模，且有效、可靠，特别在计算状态空间的性能方面，比传统建模方法更具有一定的优势。

参考文献：

[1] ARKIN A, ASKARY S, BLOCH B, et al. Web services business process execution language version 2.0 [EB/OL]. [2010-9-2]. http://www.oasis-open.org/committees/download.php/12791/.

[2] TAN W, FAN Y S, ZHOU M C. A Petri net-based method for compatibility analysis and composition of Web services in business process execution language [J]. IEEE Transactions of Automation Science and Engineering, 2009, 6(1): 94-106.

[3] 舒斌，殷国富，戈鹏，等. 基于知识的敏捷工作流系统建模方法的研究 [J]. 西安交通大学学报，2002，36(7)：731-735.
SHU Bin, YIN Guofu, GE Peng, et al. Research on modeling method for agile workflow system based on knowledge [J]. Journal of Xi'an Jiaotong University, 2002, 36(7): 731-735.

[4] 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.

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.

[5] LOHMANN N. A feature-complete Petri net semantics for WS-BPEL 2.0 [C]// 4th International Workshop on Web Services and Formal Methods. Berlin, Germany: Springer-Verlag, 2007: 77-91.

[6] STAHL C. A Petri net semantics for BPEL, Technical report 188 [R]. Berlin, Germany: Humboldt-Universit at zu Berlin, 2005: 1-84.

[7] HINZ S, SCHMIDT K, STAHL C. Transforming BPEL to Petri nets [C]// Proceedings of 3rd International Conference on Business Process Management. Berlin, Germany: Springer-Verlag, 2005: 220-235.

[8] OUYANG C, VERBEEK E, VAN DER AALST W M P, et al. Formal semantics and analysis of control flow in WS-BPEL [J]. Science of Computer Programming, 2007, 67(2/3): 162-198.

[9] YANG Y, TAN Q, YU J, et al. Transformation BPEL to CP-nets for verifying web services composition [C]// Proceedings of the International Conference on Next Generation Web Services Practices. Los Alamitos, USA: IEEE Computer Society, 2005: 137-142.

[10] 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.

[11] 门鹏，段振华. 基于着色 Petri 网的 BPEL 建模与验证 [J]. 西北大学学报：自然科学版，2007，37(6)：986-990.
MEN Peng, DUAN Zhenhua. A colored Petri net based on approach for BPEL modeling and verification [J]. Journal of Northwest University: Natural Science Edition, 2007, 37(6): 986-990.

[12] 蒋曹清，应时，文静，等. 面向服务软件异常处理过程的可终止性验证 [J]. 计算机科学与探索，2012，6(3)：208-220.
JIANG Caoqing, YING Shi, WEN Jing, et al. Verification of termination for exception handling process in service-oriented software [J]. Journal of Frontiers of Computer Science and Technology, 2012, 6(3): 208-220.

[13] CONSOLE L, FUGINI M. WS-DIAMOND: an approach to Web service-diagnosability, monitoring, and diagnosis [C]// Proceedings of the International e-Challenges Conference. Amsterdam, Netherlands: IOS Press, 2007: 105-112.

（编辑 苗凌 赵大良）

（上接第104页）

[C]// Proceeding of the first ACM International Workshop on Complex Networks Meet Information and Knowledge Management. New York, USA: ACM, 2009: 47-54.

[6] YUAN Weiwei, GUAN Donghai, LEE Y K, et al. Improved trust-aware recommender system using small-worldness of trust networks [J]. Knowledge-Based Systems, 2010, 23(3): 232-238.

[7] RUFFO G, SCHIFANELLA R. A peer-to-peer recommender system based on spontaneous affinities [J]. ACM Transactions on Internet Technology, 2009, 9 (1): 1-34.

[8] LI Yung-ming, CHEN Ching-wen. A synthetical approach for blog recommendation: Combining trust, social relation, and semantic analysis [J]. Expert Systems with Applications, 2009, 36(3): 6536-6547.

[9] KWON K, CHO J, PARK Y. Multidimensional credibility model for neighbor selection in collaborative recommendation [J]. Expert Systems with Applications, 2009, 36(3): 7114-7122.

[10] SINHA R, SWEARINGEN K. Comparing recommendations made by online systems and friends [EB/OL]. [2012-06-20] http://pdf.aminer.org/000/148/586/comparing_recommendations_made_by_online_systems_and_friends.

[11] HERLOCKER J L, KONSTAN J A, BORCHERS A, et al. An algorithmic framework for performing collaborative filtering [C]// Proceedings of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval. New York, USA: ACM, 1999: 230-237.

[12] MASSA P, AVESANI P. Trust-aware bootstrapping of recommender systems [EB/OL]. [2012-06-20] http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.108.8103&rep=rep1&type=pdf.

[13] HERLOCKER J L, KONSTAN J A, TERVEEN L G, et al. Evaluating collaborative filtering recommender systems [J]. ACM Transactions on Information Systems, 2004, 22(1): 5-53.

（编辑 武红江）