面向服务软件中基于着色 Petri 网的异常处理模型

吴 青 应 时 贾向阳
(武汉大学软件工程国家重点实验室 武汉 430072)

摘 要 面向服务软件异常处理的开发工作量大且程序逻辑复杂。若只在设计阶段后期或是编码阶段考虑异常处理功能的开发,则可能由于没有充分、完整、系统地考虑待处理的异常而严重影响软件整体可靠性。针对面向服务软件中异常处理机制的特点,提出一种基于着色 Petri 网的异常处理模型。通过对异常处理组成元素和异常处理模式分别进行形式化描述,可提供可重用的异常处理模型元素。设计者根据面向服务软件中异常处理的实际需求,使用异常处理模式连接异常处理组成元素,形成一个完整的异常处理模型。利用该模型可以精确描述面向服务软件中异常处理总体方案,便于辅助设计人员检测异常处理设计方案中的缺陷。

关键词 面向服务软件,异常处理,着色 Petri 网
中图法分类号 TP311.5 **文献标识码** A

Exception Handling Model Based on Colored Petri Net in Service-oriented Software

WU Qing YING Shi JIA Xiang-yang
(The State Key Lab of Software Engineering, Wuhan University, Wuhan 430072, China)

Abstract The workload of exception handling development is heavy, with complex program logic. If how to handle exception is considered just in the late design phase or coding phase, the omitted exceptions can seriously affect the overall reliability of the software. For the features of exception handling in service-oriented software, it presented an exception handling model based on colored Petri net. It provides reusable exception handling model elements, by formally describing the elements of exception handling and exception handling patterns. Designer connects the elements of exception handling using exception handling patterns, which forms a complete exception handling model. The model can accurately describe the overall program of exception handling in service-oriented software, aiding to detect the defects in it.

Keywords Service-oriented software, Exception handling, Colored petri nets

1 引言

异常处理设施已成为高级程序设计语言中不可或缺的部分[1],为开发高可靠的软件系统提供强有力的支持。根据 Sinha 和 Robilard 等人[2]的统计表明,异常处理代码约占总代码量的 10%。然而异常处理并未引起程序设计人员的足够重视,通常在设计阶段后期或是编码阶段才被设计人员考虑[3],这使得异常处理逻辑比正常业务逻辑更容易出错。而面向服务软件与传统软件相比,由于运行环境的动态性和不确定性以及服务资源的自治性和松耦合性,其异常处理的开发工作量更大且程序逻辑更复杂[4]。若面向服务软件中异常处理功能的开发缺乏总体规划,则异常处理难以被充分、完整、系统地考虑,可能导致由于软件中异常处理机制不完备而影响软件的整体可靠性。

针对上述问题,本文提出一种基于着色 Petri 网的异常处理形式化模型,以精确描述面向服务软件中异常处理逻辑的总体方案。首先,从面向服务软件中异常产生层次和异常处理模式两方面分析异常处理机制的特点。接着,对异常处理模型的主要成分:异常处理组成元素和异常处理模式分别给出相应的形式化模型,以便设计异常处理逻辑时复用。最后通过实例,展示"汽车装配流水线管理系统"中的异常处理模型。

2 面向服务软件中异常处理的特点

面向服务软件中异常处理机制与传统软件中异常处理机制一样,都是程序执行过程中异常产生或异常处理结束后,控制程序执行顺序的机制。需要显式或隐式地定义异常源、异常处理器和异常返回点。异常源又可以称为保护区。异常源中发生异常的点则称为异常引发点,一个异常源可以包含一个或多个异常引发点。每个异常处理器都与一个异常源相关联,响应该异常源中产生的所有异常。异常处理机制可以是语言本身固有的,也可以通过语言提供的扩展机制增加到语言上,其目的是保障软件系统的可靠性。

但是由于面向服务软件的特殊性,导致其异常处理机制与传统软件中的异常处理机制略有不同。下面分别从异常产生层次和异常处理模式两方面,阐述面向服务软件中异常处

到稿日期:2010-08-25 返修日期:2010-11-11 本文受国家自然科学基金项目(61070012)资助。
吴 青(1982—),女,博士生,主要研究方向为面向服务软件集成、Petri 网应用,E-mail:12capricorn@163.com;应 时(1965—),男,博士,教授,博士生导师,主要研究方向为面向对象软件工程方法、基于组件的软件工程方法、软件体系结构和模式、软件的可重用性与互操作性等。

理机制的特点。

基于异常产生层次，将面向服务软件中的异常分为服务异常和流程异常两大类型。服务异常是指服务被调用时所抛出的异常，将服务看成黑盒，服务异常即服务的外部异常。流程异常是指在服务协同过程中产生的异常，将流程看成黑盒，流程异常即流程的内部异常。上述两类异常的异常处理机制在异常源和异常返回点的设置上存在一定差异。①相对异常源而言：服务层的异常只由服务自身产生，一个服务就是一个异常源；而流程层的异常可能产生于该流程调用的服务、该流程调用的子流程或者该流程自身等多个位置，即一个流程可以存在一个或多个异常源。②相对异常返回点而言：服务异常经过异常处理操作后，无论返回的是正常消息还是异常消息，都将作为服务的返回消息返回给该服务的调用者。而流程异常经过异常处理操作后，若返回正常消息，则将此消息传递给紧跟该异常源的后继区域，流程继续执行，若返回异常消息，则中断该流程的执行，将此异常消息作为流程异常直接抛出。

异常处理模式逐渐成为研究工作的热点。在工作流管理系统领域，Van Der Aalst WMP 等人将工作流异常处理分为工作项级异常处理和案例级异常处理[5]，共 18 种异常处理模式。Lerner B S 在前人研究工作的基础上，较好地总结了面向服务软件中异常处理的特点，抽象出 8 种异常处理模式[6]。本文基于 Lerner B S 的研究工作，对其中常用的 5 种异常处理模式进行形式化描述。①常规模式，指为服务异常和流程异常分别定义相应的异常处理器和异常返回点。②冗余备份模式，指存在多条可供选择的路径，当其中一条路径出错时，可以选择其他路径替代执行。③日志记录模式，指异常发生时只需记录异常相关信息。④重做模式，指系统上下更新后从开始状态重新执行。⑤拒绝模式，指发生的异常过于严重导致系统直接跳转到结束状态。上述异常处理模式可以根据需要组合使用。

3 基于着色 Petri 网的异常处理模型

3.1 异常处理组成元素

异常处理组成元素由服务层异常处理组成元素 *elem_s* 和流程层异常处理组成元素 *elem_p* 组成，分别记录了异常源、异常处理器以及异常返回点。本节先定义异常处理组成元素中使用的颜色集，然后分别给出 *elem_s* 和 *elem_p* 的定义。

定义 1 Σ(颜色集)定义为：

颜色 Num=int；

颜色 $Place$= string；

颜色 $CaseStatus$=with $N|E$；

颜色 $Exception$=with none|$e1|e2|e3|e4|e5|e6$；

颜色 $Case$=record id:Num * $status$:$CaseStatus$ * at:$Place$ * $excp$:$Exception$；

颜色 $Place\times Exception$= $product\ Place * Exception$。

颜色 *Place* 为操作信息，包括流程中的活动信息或服务信息，简化为字符串。颜色 *Exception* 为枚举类型的异常类型，取值为 none 时代表无异常。颜色 *Case* 代表特定的流程实例或服务实例，由 4 个部分组成。*Id* 作为该实例的唯一标识符；*status* 表示实例当前是正常状态 N 还是异常状态 E；*at* 记录相关的操作信息：若 $status=N$，*at* 记录当前执行的操作，若 $status=E$，则记录产生异常的操作；*excp* 记录异常相关信息：若 $status=N$，*excp* 为 *none*，若 $status=E$，*excp* 记录抛出的异常类型。颜色 $Place\times Exception$ 代表操作 *Place* 可能产生的异常类型。若未作特别说明，本文规定以下异常处理组成元素的颜色集都取Σ。

3.1.1 服务层异常处理组成元素

定义 2(服务层异常处理组成元素) $elem_s=(Source_S, Handle_S, ReturnPoint_S)$是一个三元组，其中：

Source_S 代表异常源，具体见定义 3；

Handle_S 代表异常处理器，$Handle\_S=\{handle\}$，$handle\in T$；

ReturnPoint_S 代表异常返回点，$ReturnPoint\_S=\{Response\}$，$Response\in P$，也是异常源中代表消息返回位置的库所。

定义 3 *Source_S* 是一个着色网(见图 1)，$Source\_S=(\Sigma, P, T, F, C, G, E, I)$，其中：

变量：x:*Case*

库所集：$P=\{Request, Response, Storage, Outcome, ESource\}$

变迁集：$T=\{input, output, throw\}$

弧集：$F=\{\langle Request, input\rangle, \langle input, Storage\rangle, \langle Storage, active\rangle, \langle active, outcome\rangle, \langle Outcome, output\rangle, \langle output, Response\rangle, \langle Response, throw\rangle, \langle throw, Esource\rangle\}$

颜色函数集：$C=\{C(p)=Case, p\in P\}$

弧函数集：$E=\{E(f)=x, f\in F\}$

服务调用者被抽象为 *Request* 和 *Response* 库所，分别用于建模服务请求位置和消息返回位置。服务调用和消息返回行为分别建模成 *input* 和 *output* 变迁，服务执行被抽象为 *active* 变迁，*Storage* 和 *Outcome* 分别代表服务的输入输出库所。设 *Source_S* 中流动的变量类型始终是记录操作活动信息和异常类型的复合颜色 *Case*，变迁操作只改变变量的具体值，以记录当前实例的状态。故下文若无特殊说明，设变量 x 属于 *Case* 类型，颜色函数均为 *Case*，弧函数均为 x。

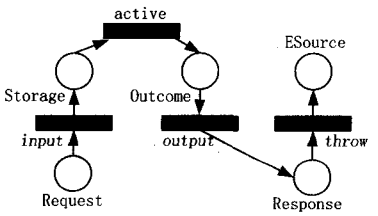

图 1 Source_S

在 *Source_S* 中，包括若干异常引发点和一个异常检测点。异常引发点指产生异常的位置。服务调用变迁 *input*，服务执行变迁 *active* 和返回消息变迁 *output* 均可能抛出异常，都是异常引发点。可以用另一个 Petri 子网，即使用替代变迁，进一步描述其可能产生的异常类型与异常发生概率。异常引发点中产生的异常只有传播到异常检测点后，才能被异常处理器接获并处理，*Esource* 为 *Source_S* 中的异常检测点。

3.1.2 流程层异常处理组成元素

定义 4(流程层异常处理组成元素)

$elem_p=(Source_P, Handle_P, ReturnPoint_P)$是一个三元组，其中：

Source_P 代表异常源，具体见定义 6；

Handle_P 代表异常处理器，具体见定义 7；

ReturnPoint_P 代表异常返回点，*ReturnPoint_S*={*ReturnPoint*, *ProcessEnd*}，*returnPoint*, *ProcessEnd*∈*P*

流程层异常处理组成元素中的异常源是指流程中包含多个子活动的一个结构活动，所有子活动产生的异常都将抛给该结构活动。包含两个串联子活动的流程异常源模型如图 2 所示，其中虚线框外部的结构是异常源的外部固定结构，虚线框内部包含各种子活动集。

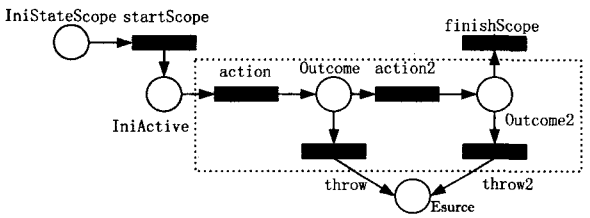

图 2　Source_P

定义 5　*Source_P*=(∑, *P*, *T*, *A*, *F*, *C*, *G*, *E*)，其中：

库所集：*P*={*iniStartScope*, *ESource*}

变迁集：*T*={*startScope*, *finishScope*}

子活动集：*A* 是一个着色 Petri 网

弧集：*F*={⟨*IniStartScope*, *startScope*⟩, ⟨*startScope*, *A*. *IniActive*⟩, ⟨*A*. *output*, *fnishScope*⟩, ⟨*A*. *throw*, *ESource*⟩}

值得注意的是，相邻的 *Source_P* 通过 *iniStartScope* 库所连接。子活动集 *A* 中包含的任意子活动均可作为异常引发点抛出相应的异常。*ESource* 是异常检测点，接收 *Source_P* 中产生的所有异常。

定义 6　*Handle_P* 是一个着色 Petri 网(见图 3)，*Handle_P*=(∑, *P*, *T*, *F*, *C*, *G*, *E*, *I*)，其中：

库所集：*P*={*Judgement*}

变迁集：*T*={*handle*, *succeed*, *fail*}

弧集：*F*={⟨*handle*, *Judgement*⟩, ⟨*Judgement*, *succeed*⟩, ⟨*Judgement*, *fail*⟩}

哨函数：*G*(*succeed*)=(*x*. *status*=*N*)，*G*(*fail*)=(*x*. *status*=*E*)

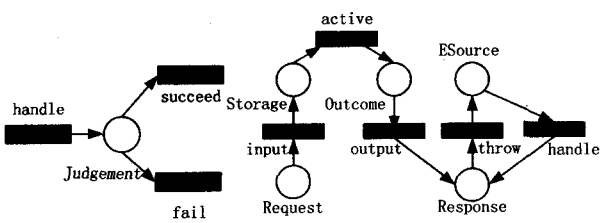

图 3　Handle_P

图 4　处理服务异常的常规模式

3.2　异常处理模式

异常处理模式用于描述服务层异常处理组成元素 *elem_s* 和流程层异常处理组成元素 *elem_p* 中异常源、异常处理器和异常返回点之间的连接关系。

(1)常规模式

常规模式是最常见的异常处理模式，处理服务异常的常规模式如图 4 所示。处理流程异常的常规模式如图 5 所示。

设服务层异常处理组成元素 *elem_s*，其常规模式如图 5 所示，其中：

F={⟨*ESource*, *handle*⟩, ⟨*handle*, *Response*⟩}

G(*throw*)=(*x*. *status*=*E*)

设流程层异常处理组成元素 *elem_p*，其后继区域的初始库所为 IniStartScope2，*elem_p* 所在流程的终止库所为 *ProcessEnd*，其常规模式如图 5 所示，其中：

F={⟨*ESource*, *handle*⟩, ⟨*succeed*, *IniStartScope2*⟩, ⟨*fail*, *ProcessEnd*⟩}

G(*throw*)=*G*(*handle*)=*G*(*fail*)=(*x*. *status*=*E*)

G(*succeed*)=(*x*. *status*=*N*)

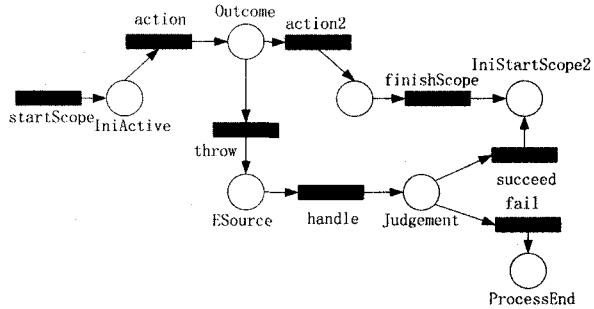

图 5　处理流程异常的常规模式

(2)冗余备份模式

冗余备份模式适用于具有多条并行路径的流程，这些并行路径的执行是无序的，任意路径成功执行都代表该流程成功完成。当运行的路径抛出异常时，异常处理操作记录下该路径的信息，以便重试时流程可以选择其他可执行路径，如图 6 所示。

设流程层异常处理组成元素 *elem_p*，其中，*IniA* 和 *IniB* 分别代表冗余路径的开始库所。

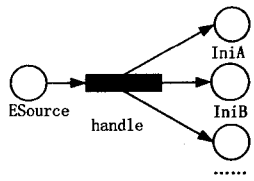

图 6　冗余备份模式

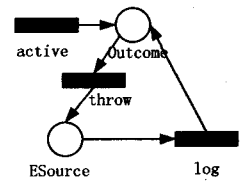

图 7　日志记录模式

F={⟨*Esource*, *handle*⟩, ⟨*handle*, *IniA*⟩, ⟨*handle*, *IniB*⟩}

G(*handle*)=(*x*. *status*=*E*)

(3)日志记录模式

日志记录模式适用于产生的异常不足以影响实例运行或需要积累更多的异常信息方可进行异常处理，异常处理操作记录下相关异常信息，系统继续执行，让用户根据需要自行选择处理异常的时间和方式，如图 7 所示，其中：

F={⟨*Esource*, *log*⟩, ⟨*log*, *Outcome*⟩}

G(*throw*)=*G*(*log*)=(*x*. *status*=*E*)

(4)重做模式

重做模式适用于抛出的异常对系统没有任何影响，系统只需刷新上下文信息后，重新执行即可。例如，用户没有订购到某日的机票，可以选择订购其他时间的机票来处理该异常，如图 8 所示。

F={⟨*ESource*, *handle*⟩, ⟨*handle*, *IniActive*⟩}

G(*throw*)=(*x*. *status*=*E*)

(5)拒绝模式

拒绝模式适用于抛出及其严重的异常，导致系统直接结束。例如，用户申请银行信贷业务时，系统在检测时发现该用

户没有足够的信贷能力，于是采用拒绝的方式结束执行。如图 9 所示，其中，设系统终止位置为 *Final* 库所。

$F=\{\langle ESource, reject\rangle, \langle reject, Final\rangle\}$

$G(throw)=G(reject)=(x.status=E)$

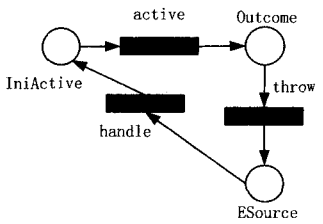

图 8　重做模式

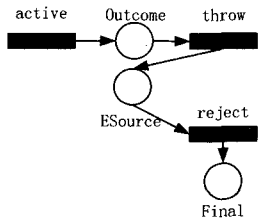

图 9　拒绝模式

4　案例研究

下面给出汽车装配流水线管理系统（简称 R2E 系统）中部分功能。假设汽车车架到达流水线中安装车轮的工位，R2E 系统需要获取待安装的物料信息，即获取剩余车轮的数目。检查流水线中用于执行安装的设备状态，选择工作状态良好且当前状态为空闲的安装设备。当安装工作完成后，R2E 系统还需获取安装到该车架上的车轮条码。如表 1 所列，上述步骤在实际使用中可能出现多种异常情况。由此可见此例中异常处理逻辑远比正常处理逻辑复杂得多。

分析可知，正常业务流程中的 3 大步骤是顺序依次相连。情况①使用日志备份模式处理，情况②适用于拒绝模式，情况③适合冗余备份模式来处理，常规模式来处理情况④和⑤，情况⑥采用重做模式。

运用着色 Petri 网工具 Design/CPN 构建 R2E 系统的异常处理模型如图 10 所示。该流程存在异常源 *Source_P1* 和 *Source_P2*，其中 *Source_P1* 包含获取车轮数目变迁 *cNum* 和检查设备状态变迁 *sta1* 或 *sta2*，*Source_P2* 包含获取条码变迁 *col*。异常处理变迁 *h_i* 对应处理表 1 中第 *i* 种异常情况。

表 1　R2E 系统中可能出现的异常情况

操作步骤	异常情况	异常处理
获取物料数量	①数量低于阈值	在日志中记录
	②数量为零	拒绝后续安装
检查设备状态	③设备忙碌	选择另一台设备
	④设备无应答	检查使用的 web 服务
获取安装的车轮条码	⑤车轮与车架型号不匹配但通用	不处理
	⑥未获取到车轮条码，车轮被漏装	重新安装车轮

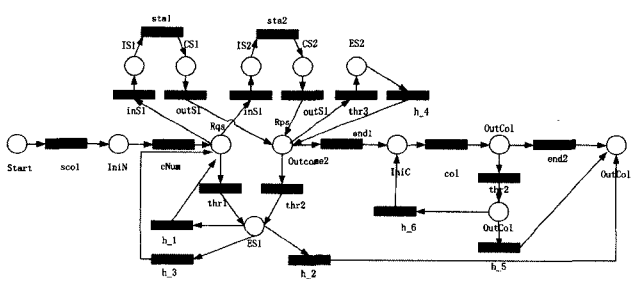

图 10　R2E 系统中的异常处理模型

5　相关工作

目前，学术界通常基于正常业务逻辑和异常处理逻辑分离的思想，使用 CSP、B 方法或 Petri 网形式化规约异常处理逻辑。

CSP 和 B 方法是一种较好的用于实时控制系统构架分析的形式化方法。英国 Newcastle 大学的 Alexander Romanovsky 等人利用协调原子动作（Coordinated Atomic Actions）来表示和建模并发系统中的异常处理机制，使用 B 方法和 Alloy 语言对基于协调原子动作所设计的并发系统的结构以及动作之间异常流相关的信息进行规约[7]。巴西 São Paulo 大学的 Pereira 等人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[8]，用于指导并发容错系统的形式化规约。该模型提供一组 CSP 进程和预定义通道来规约和协调组件的异常处理。北京大学的裘宗燕等人定义了基于 CSP 的 PPL 语言[9]来形式化描述协同异常处理过程的操作语义则。巴西 Campinas 州立大学的 Patrick H. S. Brito 等人提出了一种基于 B 方法和 CSP 分析和验证容错软件体系结构异常传播的方法[10]。但由于 CSP 是基于事件进程的，在建模能力、数据抽象方面有一定缺陷，而 B 方法在描述系统并发控制等方面又存在局限。因此，使用上述方法规约软件系统中的异常处理逻辑存在一定的不足。

美国 Carnegie Mellon 大学软件工程研究所的 Peter Feiler 等人利用基于 Petri 网的错误模型 Annex[11]，描述各种体系结构构件的错误状态、错误状态迁移及触发错误状态迁移的错误事件（异常情况）和错误传播，并对它们进行有选择的过滤和屏蔽，以制定相应的容错策略。澳大利亚新南威尔士大学的 Rachid Hamadi 通过扩展 Petri 网提出了一种自适应恢复网（SARN），用于设计时描述工作流的异常行为[12]，通过定义基于任务和基于区域的两种恢复策略，SARN 既可以处理一般的预定义的恢复策略，也可以处理用户自定义恢复策略。南京大学窦万春、蔡士杰等人基于扩展 Petri 网所定义的过程单元，构造了面向意外处理的、集成控制流逻辑和数据流逻辑的工作流模型[13]。华东理工大学的虞慧群、范贵生等人利用 Petri 网对服务组合中不同的故障情况进行建模，并依据组件的关系生成服务组合的故障处理模型[14]。但上述研究工作并未完整考虑面向服务软件的特殊性，如缺乏对流程中异常保护区的建模，也没有提供复用机制来重用已经建立好的模型。

本文基于 Petri 网可以广泛应用于描述和研究并发、异步和分布式特征系统的特点提出了一种面向服务软件中的异常处理模型。该模型区分服务层和流程层异常处理机制的差异，且融入异常处理模式，便于设计人员快速复用已有的异常处理方案。

结束语　本文首先从异常产生层次和异常处理模式两个方面，阐述面向服务软件中异常处理机制的特点。基于上述特点，本文提出一种基于着色 Petri 网的异常处理模型。该模型将异常处理模式视作可复用的异常处理流程结构，连接服务层和流程层中各类异常处理组成元素（如异常源、异常处理器和异常返回点），形成一个完整的异常处理模型。该模型可以有效地描述一个高抽象层次的、易于理解的、易于验证评估的异常处理总体方案。如何基于该模型验证异常处理逻辑的正确性和可终止性，以辅助设计人员检测异常处理设计方案中的缺陷，将是我们下一步研究工作的重点。

参 考 文 献

[1]　Jiang S J, Xu B W. Exception handling—An approach to impro-

ving software robustness[J]. Computer Science, 2003, 30(9): 169-172

[2] Illard R M P, Murphy G C. Static analysis to support the evolution of exception structure in objec-oriented systems [J]. ACM Transactions on Software Engand Methodology, 2003, 12(2): 1912-1921

[3] Lemos R D, Romanovsky A. Exception handling in the software lifecycle[J]. International Journal of Computer Systems Science and Engineering, 2001, 16(2): 167-181

[4] Chan K S M, Bishop J, Steyn J, et al. A Fault Taxonomy for Web Service Composition[C]// Proceeding of International Conference on Service-Oriented Computing (ICSOC 07). Springer, 2007: 363-375

[5] Russell N, van der Aalst W, ter Hofstede A. Workflow exception patterns[C]// Proceeding of 18th International Conference on Advanced Information Systems Engineering (ICAISE06). Springer, 2006: 288-302

[6] Lerner B S, Christov S, Osterweil L J, et al. Exception Handling Patterns for Process Modeling[J]. IEEE Transactions On Software Engineering, 2010, 36(2): 162-183

[7] Castor F F, Romanovsky A, et al. Improving reliability of cooperative concurrent systems with exception flow analysis[J]. Journal of Systems and Software, 2009, 82(5): 874-890

[8] 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

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

[10] Brito P H, de Lemos R, Rubira C M, et al, Architecting Fault Tolerance with Exception Handling: Verification and Validation [J]. Journal of Computer Science and Technology, 2009, 24(2): 212-237

[11] Feiler P, Rugina A. Dependability Modeling with the Architecture Analysis & Design Language (AADL) [R]. CMU/SEI-2007-TN-043, 2007: 1-76

[12] 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

[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

[14] 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

(上接第 132 页)

上的优越性。当链路质量相对较好时，Rt-Rel 方案在 $n=2$ 和 $n=3$ 时的 CANum 接近，这是因为大多数呼叫连接可由第一条路径来完成；当链路质量显著下降时，$n=2$ 时的 CANum 要小于 $n=3$ 时的，原因在于不佳的链路质量使得尝试次数增加，后继的路径将更多被采用，路集中后继路径的差异性的影响开始显现。

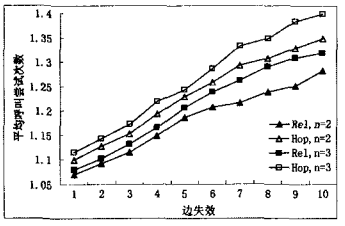

图 3　不同路集选择方法的平均呼叫尝试次数比较

结束语　本文对 SIP 呼叫连接消息的选路问题进行了研究，提出了用于描述一点对多点中任意一点的可靠度计算模型，以及用于优化呼叫成功率和呼叫连通速度的选路方法。文中给出了该方法的理论分析和比较，以及验证方法有效性的仿真实验。文中提出的 n 次最可靠 $1/K$ 呼叫路集能提供最高的 n 次尝试内总呼叫成功率，同时在每次尝试中优先选择成功率最高的路径，以实现最快的连通。随着最大尝试次数的增加，总呼叫成功率会提升，但是呼叫时延及呼叫尝试次数会增加，因而需要在网络链路质量不佳时仔细地权衡选择最大呼叫尝试次数，以实现两个性能指标的折中。

深入的工作包括应对网络拓扑变化的周期式路由更新策略，并有必要探讨如何将多个 SIP 服务器按 P2P 的方式优化组织，为大型无线 Mesh 网(如含有几百个无线 MR 的城域 Mesh 网)提供高可靠度的连接、高效的资源发布以及消息的转发。

参 考 文 献

[1] Rong Bo, Qian Yi, Chen Hsiao-hwa. An enhanced SIP proxy server for wireless VoIP in wireless mesh networks[J]. IEEE Communications Magazine, 2008, 46(1): 108-113

[2] Akyildiz I F, Wang Xu-dong. A survey on wireless mesh networks[J]. IEEE Communications Magazine, 2005, 43(9): 23-30

[3] Hsu Chun-yen, Wu Jean-lien C, Wang Shun-te, et al. Survivable and delay-guaranteed backbone wireless mesh network design [J]. Journal of Parallel and Distributed Computing, 2008, 68 (3): 306-320

[4] De Couto Douglas S J, Aguayo D, Bicket J, et al. A high-throughput path metric for multi-hop wireless routing[J]. Wireless Networks, 2005, 11(4): 419-434

[5] Angelos V, Kong L L, Ioannis B, et al. Assessing link quality in IEEE 802.11 wireless networks: Which is the right metric? [C]// Proceedings of 19th IEEE International Symposium on Personal, Indoor and Mobile Radio Communications. 2008

[6] 何明，裘杭萍，胡爱群，等. Mesh 网的可靠性评价方法研究[J]. 东南大学学报：自然科学版，2008，38(SUPPL. 1)：222-225

[7] Egeland G, Engelstad P E. The Availability and Reliability of Wireless Multi-Hop Networks with Stochastic Link Failures [J]. IEEE Journal on Selected Areas in Communicaions, 2009, 27(7): 1132-1146

[8] 赵蕴龙，单宝龙，高振国，等. 无线 Mesh 网骨干层 2-终端可靠性计算策略[J]. 计算机学报，2009，32(3)：424-431

[9] Zhao Lian-chang, Kong Fan-jia. New formula and an algorithm for reliability analysis of networks[J]. Microelectronics Reliability, 1997, 37(3): 511-518

[10] Weh Wei-chang. A greedy branch-and-bound inclusion-exclusion algorithm for calculating the exact multi-state network reliability[J]. IEEE Transactions on Reliability, 2008, 57(1): 88-93