一种容错的面向服务体系结构描述语言

吕国斌 王权于 李贵龙
(中国地质大学(武汉)远程与继续教育学院 武汉 430074)

摘 要 针对如何构建具有容错能力的面向服务软件体系结构的科学问题,提出了一种新型的支持异常处理的面向服务体系结构描述语言SOADL-EH。该语言不仅具备表示面向服务软件体系结构的建模能力,还提供了异常处理服务、异常连接件及异常配置等语言成分,清晰地将面向服务软件体系结构层的异常处理逻辑从正常业务逻辑中分离出来,弥补了已有的面向服务体系结构描述语言在建模异常处理能力方面的不足。

关键词 面向服务软件体系结构,体系结构描述语言,异常处理

Fault-tolerant Service-oriented Architecture Description Language Supporting Exception Handling

LU Guo-bin WANG Quan-yu LI Gui-long
(College of Learning and Continuing Education, China University of Geoscience, Wuhan 430074, China)

Abstract To solve the problem of the construction of fault tolerant service-oriented software architecture, the paper proposed SOADL-EH, a new service-oriented software architecture description language (ADL) supporting exception handling. With the capability of modeling normal SOA and the provision of exception handling services(EHService), exception handling connectors(EHConnector) and exception handling configurations(EHConfigure), etc. software architecture elements, this language can separate clearly the exception handling logic from the normal business logic of the service-oriented software architecture layer, which covers the shortage in exception handling capability of the existing ADLs.

Keywords SOA, ADL, Exception handling

1 引言

面向服务是一种新兴的软件开发范型[1],服务资源的自治性和松耦合性、面向服务软件系统运行环境的动态性和不确定性,都给面向服务软件系统的可用性和可信性带来了严峻的挑战[2,3]。目前,由于面向服务软件开发技术并未在软件开发的早期对异常处理的规约和设计提供足够的支持,因此造成异常处理功能的开发缺乏总体规划,异常处理难以被充分、完整和系统地理解。因此,对于构建高可信的面向服务软件系统而言,在软件开发的早期阶段,特别是体系结构设计阶段,对异常及其处理进行规约和设计是十分必要的。

在软件体系结构的研究中,通常使用体系结构描述语言(Architecture Description Language, ADL)来对体系结构建模。目前主流的ADL有Union, Wright, C2, Rapide, ACME, xADL和πADL。文献[4]给出了目前主流ADL的分类和比较框架,该框架明确地规约了构件、连接件和体系结构配置的建模能力。这些ADL强调了体系结构的不同侧面,对体系结构的研究起到重要的作用。面向服务软件体系结构(Service-Oriented Architecture, SOA)是一种针对互联网的新型的分布式软件体系结构[5]。当前针对SOA的ADL的研究工作主要是对已有的基于分布式的体系结构描述语言的扩展[6],其不足在于缺乏对SOA自身特点和面向服务开发中的特殊要求的支持,使得体系结构设计无法指导面向服务的软件系统设计和实现。虽然,近几年也有部分研究工作开始针对面向服务的软件体系结构特点和开发需求,提出了一些新颖的面向服务体系结构描述语言[7-10],但这些体系结构描述语言往往只关注面向服务软件体系结构的正常逻辑建模能力,对于面向服务软件体系结构的异常及其处理的建模能力还缺乏足够的语言设施支持。

因此,结合面向服务软件体系结构特点和开发需求,设计一种新型的支持异常处理的面向服务体系结构描述语言,清晰地将面向服务软件体系结构层的异常处理逻辑从正常业务逻辑中分离出来,完善已有的ADL在描述面向服务体系结构异常处理方面的不足,对于构建高可信的面向服务软件系统而言是十分重要的。

本文第2节介绍相关研究工作;第3节介绍SOADL-EH语言的设计方案;第4节定义SOADL-EH的语法结构;最后是总结和对未来工作的展望。

2 相关研究工作

与本文相关的研究工作主要涉及到以下两个方面:①面向服务的体系结构描述语言:采用哪些体系结构元素来表示面向服务软件系统的体系结构;②软件体系结构层的异常处理机制:如何将异常处理技术融入到软件系统的体系结构设

到稿日期:2010-03-20 返修日期:2010-07-02 本文受湖北省自然科学基金项目(2009CDB307)资助。
吕国斌(1965-),教授,主要研究方向为软件工程、网络安全,E-mail:gblv@cug.edu.cn;王权于(1971-),副教授;李贵龙(1962-),工程师。

计中，构建具有容错能力的软件体系结构。下面分别从这两个方面对国内外的相关研究工作进行分析和总结。

2.1 面向服务体系结构描述语言相关研究

随着面向服务开发技术的发展，国内外学术界和工业界相继提出了一些面向服务的 ADL，这些 ADL 在不同程度和不同方面可以支持面向服务软件系统的体系结构描述。文献[6]通过对基于 MDA 的面向服务软件系统开发方法的研究，设计了基于 MDA 的面向服务软件体系结构的 UML 模型，该模型考虑到面向服务软件系统的动态性和移动性等特点，采用体系结构描述语言 πADL 来表示和规约面向服务的软件体系结构；文献[7]提出了一种扩展 XADL 来描述面向服务体系结构的方法；文献[8]通过对 BPEL 组合的面向服务系统的研究，定义了基于 BPEL4WS 的 Web services 组合系统体系结构风格，并针对这种风格设计了软件体系结构描述语言 WSC/ADL。WSC/ADL 语言成分包含描述 Web services 的服务组件、描述 Web services 之间交互的连接器以及描述系统拓扑结构的配置，文中显式地将连接间作为第一类建模实体。文中对服务组件、连接器以及配置行为的描述同 Wright 中较为类似，同样使用了 CSP 描述组件和连接器的行为。文献[9]在对 XYZ/ADL 进行扩展的基础上，对服务、合成服务、面向服务的体系结构以及系统的活性、正确性以及安全性等核心概念进行了形式化定义。该文将服务以及服务的实现组件封装在一起建模为一个服务角色对象(ServiceRoleObject)。服务提供者、服务请求者和服务代理分别表示为 3 种不同的服务角色对象，这 3 种服务角色对象之间的交互使用了 4 种不同的连接器分别来描述发布、查找、绑定和交互 4 种通信方式。文献[10]针对 SOA 的特点，提出了面向服务的软件体系结构描述语言 SOADL，并给出了使用 SOADL 描述 SOA 的方法。SOADL 从结构、行为、设计约束几个方面描述 SOA。在结构方面，提出了基于消息的服务接口模型来描述服务的结构；在行为方面，定义了相应的行为描述机制来描述服务的个体行为，并提出了行为合成的算法来支持将服务的个体行为组合起来得到系统的全局行为，以及得到复合服务的外部行为。在设计约束方面，提出了一种约束描述方法，来支持面向服务体系结构的设计约束的描述。文献[11]提出了一个从服务规范阶段逐步设计软件体系结构的方法，采用 XML 作为元语言，并集成相关的 UML 2.0 Profile 规则、OCL 语言和 MSC 表示法，设计了一种新的基于 XML 的描述 SOA 的体系结构描述语言 S-XADL。S-XADL 定义服务为体系结构的构造块，定义服务的交互模式为响应体系结构的横切方面，并且在 ADL 层给予明确的描述。S-XADL 将角色模型抽象为一种具有交互模式的服务角色加以抽象，并作为基本元素进行规约。S-XADL 为具体构件和角色相联系提供多种手段，并以此生成一个体系结构配置。

2.2 软件体系结构层异常处理的相关研究

文献[12]中 Fernando Castor Filho 等人通过扩展 ACME，使其在风格规约中支持异常风格的描述，以表示产生异常的组件、捕获异常的组件和连接这些组件的异常管道；以及使其在系统规约中支持异常流视图的描述。美国 Carnegie Mellon 大学软件工程研究所的 Peter Feiler 等人研究了具有容错能力的嵌入式系统软件体系结构的建模方法[13]。该方法在 AADL 模型的基础上，利用错误模型 Annex，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。文献[14]在 C2 软件体系结构风格的基础上，定义了 iC2C 组件(idealized C2 Component)，并将体系结构元素的组件划分为处理正常行为和处理异常行为两个部分，以清晰地描述和分析基于异常处理的容错软件系统体系结构。英国 Kent 大学的 R. de Lemos 等人在 iC2C 组件研究基础上，进一步地扩展定义了容错体系结构元素 IFTE(The idealized fault-tolerant element)，并基于异常处理技术，将 IFTE 清晰地分隔成正常部分和异常部分，以描述软件体系结构的异常检测和异常处理[15]。

上述这些工作在本文的研究工作中都有借鉴。虽然许多研究机构在异常处理方面开展了多年的研究，也取得了一些有价值的研究成果，但在研究支持异常处理的面向服务软件体系结构描述语言方面，目前还存在着以下的问题：①面向服务软件异常处理机制的研究主要集中在软件实现阶段和运行阶段，支持异常及其处理逻辑建模的 ADL 的研究相对不足。②对于软件体系结构层异常处理机制的研究，主要针对基于组件的软件体系结构进行，针对面向服务软件体系结构的研究工作相对较少。

3 SOADL-EH 设计方案

3.1 设计原则

SOADL-EH 语言设计遵循以下原则：

1)符合 ADL 的设计要求：SOADL-EH 作为一种软件体系结构描述语言，首先需要符合一个 ADL 的主要设计要求；

2)体现面向服务软件体系结构的特点：SOADL-EH 是一种面向服务的软件体系结构描述语言，SOADL-EH 的设计要针对 SOA 以及面向服务开发的特点，能够描述服务消息接口、服务的协同和交互行为、服务之间基于消息的松耦合通信方式。

3)关注点分离的原则：SOADL-EH 作为一种具有容错能力的软件体系结构描述语言，应提供充分的语言成分，能分别对面向服务软件系统的正常处理逻辑和异常处理逻辑进行建模；

4)简单性和易理解性：SOADL-EH 应尽量采用简洁、容易理解的语法和描述方式，以方便设计人员进行体系结构设计。

基于上述设计原则，参考文献[3]中的通用 ADL 分类和比较框架，针对体系结构层异常处理的特点，我们高层抽象出支持异常处理的面向服务体系结构，如图 1 所示。SOADL-EH 语言显式地将连接件定义为第一类建模实体；提供基本服务、编排服务、常规连接件和常规配置 4 种语言成分来描述面向服务软件系统的正常业务逻辑；同时提供异常处理服务、异常连接件和异常配置来描述面向服务软件系统的异常处理逻辑。

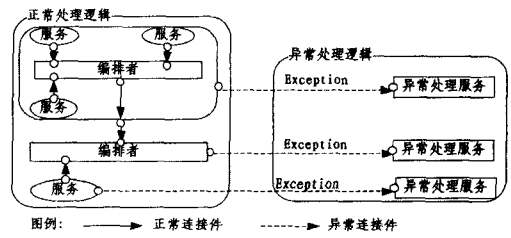

图 1 支持异常处理面向服务的软件体系结构

3.2 概念模型

SOADL-EH 语言是用来描述支持异常处理的面向服务软件体系结构的形式化符号。SOADL-EH 语言一方面遵循文献[3]给出的已被广泛认同的体系结构描述框架，围绕组件、连接件和配置等体系结构元素来描述软件体系结构；另一方面，SOADL-EH 语言将异常处理机制融入到软件体系结构的描述中，支持具有容错性的面向服务软件体系结构的表示。

基于上述关注点分离的原则，将支持异常处理的面向服务软件体系结构(Architecture)核心元素(如图 2 所示)分为常规体系结构概念和异常体系结构概念并分别描述如下。

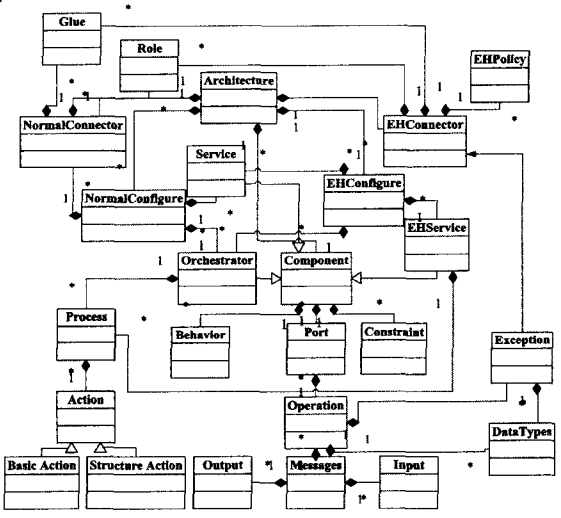

图 2 SOADL-EH 概念模型

3.2.1 常规体系结构元素

(1)基本服务(Service)：基本服务是描述正常业务逻辑的原子性的功能单元和计算实体。服务可表示为一个由服务端口、服务行为和约束的三元组：Service=(Port, Behavior, Constraint)。服务端口由一组操作实现具体的功能，每个操作(Operation)可表示为 Operation=(Input, Output, Exceptions)，即输入消息、输出消息和异常消息集合。异常(Exception)是一种特殊的消息类型定义。

(2)编排者(Orchestrator)：编排者是描述基于组合过程的复合服务组件。编排者可表示为一个四元组：Orchestrator=(Port, Structure, Process, Constraint)，即编排者对外提供的服务接口、内部组织结构、编排过程和必须遵守的约束。

(3)常规连接件(NormalConnector)：常规连接件被定义为第一类实体，描述了基本服务和编排者之间的连接和交互，表示为一个由连接角色和交互协议组成的二元组：NormalConnector=(Role, Glue)。连接角色是基本服务、编排者和常规连接件的交互点。

(4)常规配置(NormalConfigure)：描述了基本服务实例、编排者实例和常规连接件实例之间的拓扑关系。

3.2.2 异常体系结构元素

(1)异常处理服务(EHService)：异常处理服务是异常处理机制的处理器，描述了体系结构层的异常的处理过程。异常处理服务可表示为一个端口、处理过程和约束的三元组：EHService=(Port, Process, Constraint)。其中，端口表示对外提供异常处理的功能，处理过程表示如何处理接受到的异常，约束表示对异常处理服务的限制。异常处理服务端口描述方式与基本服务相同。

(2)异常连接件(EHConnector)：描述体系结构中异常的捕获和转发，定义了正常业务逻辑中的基本服务、编排者和异常处理服务之间的交互关系。异常连接件表示为 EHConnector=(Role, Glue)，即异常连接件的角色和交互协议。异常连接件的角色是基本服务、编排者与异常服务的交互点；交互协议基于一组异常处理策略(EHPolicy)来描述异常源、异常处理器和异常返回点之间的关系，异常处理策略定义为一个三元组：EHPolicy=(ESource, EHandler, EReturnPoint)，其中，ESource 表示异常源，EHandler 表示异常处理器，EReturnPoint 表示异常处理后的返回位置。

(3)异常配置(EHConfigure)：体系结构异常配置是面向服务软件系统运行时刻异常处理逻辑的快照，描述了基本服务实例、编排者实例、异常处理服务实例和异常连接件实例之间的拓扑关系。

4 SOADL-EH 语言

针对上述概念模型，我们设计了支持异常处理建模的面向服务软件体系结构描述语言 SOADL-EH。采用 BNF 范式来规约 SOADL-EH，其中，“//”表示注释，“<…>”表示非终结符，“[…]”表示出现 0 次或者多次，“[…]+”表示出现 1 次或者多次，“|”表示可选项。

SOADL-EH 语言充分体现了关注点分离原则。SOADL-EH 语言提供了基本服务、编排者、常规连接件和常规配置 4 种语言成分支持正常业务逻辑的描述，并提供了异常处理服务、异常连接件和异常配置 3 种语言成分支持面向服务软件体系结构异常处理逻辑的描述。具有异常处理能力的面向服务软件系统体系结构(Architecture)定义如下：

```
〈Architecture〉::= //软件体系结构
[〈Service〉]+ //基本服务
[〈Orchestrator〉]+//编排者
[〈NormalConnector〉]+//常规连接件
[〈NormalConfigure〉]+//常规配置
[〈EHService〉]+//异常处理服务
[〈EHConnector〉]+//异常连接件
[〈EHConfigure〉]+//异常配置
End Architecture
```

我们对 SOADL-EH 体系结构中的每种元素分别描述如下。

4.1 基本服务

基本服务(Service)是构建面向服务软件系统的基本单元。每个服务至少包含一个端口，一个行为描述以及设计约束。端口是基本服务与外部环境的交互点；行为规约了基本服务间的交互协议；约束规定了基本服务应遵守的一些限制性条件。

```
〈Service〉::= Service〈sname〉
        [〈Port〉]+
        [〈Service_Behavior〉]+
        [〈Service_Constraint〉]+
End Service
```

端口(Port)：根据端口在服务与外部环境交互中所扮演的角色，将端口分为提供者端口(Provider)和请求者端口(Requester)。提供者端口是服务向外提供功能的接口；请求

者端口则是服务使用外部功能的接口。每个端口至少包含一个操作(Operation)。操作代表了基本服务的具体功能。端口中定义了该端口所接收或发送的消息类型(messageTypes),同时还定义了该端口所要引发的异常类型(exceptionTypes)。

```
〈Port〉::= Port〈Port_name〉
        [〈PortType〉]
        [〈DataTypes〉]+
        [〈MessageTypes〉]+
        [〈ExceptionTypes〉+]
        [〈Operation〉]+
End Port
〈PortType〉::= provider | requester
```

基本数据类型(DataType):SOADL-EH 直接导入 XML Schema 来定义基本数据类型。其定义如下:

```
〈DataTypes〉::=〈XMLNameSpace_Name〉
〈XMLSchema_DataType_Name〉
```

消息类型(MessageTypes):消息包括输入消息(Input)、输出消息(Output);每条消息都可以包含若干属性。消息类型定义包括基本消息定义、复杂消息定义和导入外部消息定义 3 种方式:

```
〈MessageTypes〉::=〈Basic_MessageType_Def〉 |
        〈Complex_MessageType_Def〉 |
        〈Import_MessageType_Def〉
```

基本消息定义(Basic_Message_Def):如果消息类型只包含单个属性,且该属性为 DataTypes 中定义的基本数据类型,如整型、字符串型等,则可以在消息类型定义中直接声明对应的数据类型,定义如下:

```
〈Basic _MessageType_Def〉::= Message〈Name〉
      〈property_name〉:〈dataType_name〉
//消息包含的属性定义
End Basic_MessageType_Def
〈dataType_name〉::=〈XMLNameSpace_Name〉
〈XMLSchema_DataType_Name〉
```

复杂消息定义(Complex_MessageType_Def):如果一个消息类型有复杂的数据结构,由多个简单数据类型复合而成的。其定义如下:

```
〈Complex_MessageType_Def〉::= Message 〈Name〉 [〈property
 _name〉:〈dataType_name〉]+
End Complex_MessageType_Def
```

导入消息类型(Import_MessageType_Def):对消息类型的定义可以使用导入机制,将外部 xsd 文件导入到消息类型定义中。

```
〈messageType〉::= 〈XMLNameSpace_Name〉
      〈XMLSchema_messageType_Name〉
```

异常(Exception):异常是一类特殊的消息类型,描述了一个预定义的错误报告机制,其基本定义如下:

```
〈Exception〉::=Exception〈Exception_Name〉
        [〈Description〉]
        [〈Timestamp〉]
        [〈OriginatorReference〉]+
End Exception 其中:
Description:异常信息描述;
Timestamp:异常发生的时间;
```

OriginatorReference:产生故障的组件的端口引用;这里所给出的是异常的基本定义,可在基本定义基础上扩展对异常的定义。

操作(Operation):在面向服务体系结构中,操作涉及到在组件之间进行消息交换。根据消息交换的方向,操作有 4 种消息交换模式(MEP):in-only,out-only,in-out,out-in。In-only 模式的操作只有一个输入消息,没有输出消息;out-only 模式的操作只用一个输出消息,没有输入消息;in-out 模式的操作是先接受输入消息再发送输出消息;out-in 模式的操作是先发出输出消息再接受输入消息。其定义如下:

```
〈operation〉::=operation〈operation_Name〉
        [〈MEP〉]
        [〈message〉]+
        [〈exception〉]+
End operation
〈MEP〉::= in-only | out-only | in-out | out-in
〈message〉::=〈messagc_Name〉|
〈message_Name〉,〈message_Name〉
〈exception〉::=[〈xception_Name〉]+
```

基本服务行为(Service_Behaviour):行为描述基本服务与外部环境交互中的契约。基本服务的行为由一组动作(Action)构成。动作分为两类:基本动作(Basic Action)和结构化动作(Structured Action)。基本动作是原子的、不可分解的动作。结构化动作是由多个动作按照一定的时序关系复合而成的动作。服务行为定义如下:

```
〈Service_Behaviour〉::=Service_Behaviour〈SB_Name〉
      [Action]+
End Service_Behaviour
〈Action〉::= 〈BasicAction〉 |〈StructureAction〉
〈StructionAction〉::=StructionAction〈SAName〉
      [BasicAction]+
End StructionAction
```

基本活动定义可以采用文献[16]中基于 π 演算类似的方式来描述。

基本服务约束(Service_Constraint):描述了基本服务需要满足的设计限制,约束的描述使用一阶谓词逻辑的断言来表示,断言分为不变式(Invariant)和启发式(Heuristic),前者是强制性的约束,后者是非强制性的约束。其定义如下:

```
〈Service_Constraint〉::=Service_Constraint〈SC_name〉
  [〈assertion〉:〈Invariant〉]+ //不变式
  [〈asseration〉:〈Heuristic〉]+ //启发式
End Service_Constraint
```

4.2 编排者

编排者(Orchestrator)是一种基于编排过程的组合服务。整个面向服务的软件系统(System)也可以是一个编排者。编排者包括对外提供的服务接口、支持服务编排的内部结构、实现服务编排的过程和编排约束。

```
〈Orchestrator〉::=Orchestrator〈O_Name〉
[as System]
      [〈Port〉]+
      [〈Orchestrator_Structure〉]
      [〈Orchesrator_Behavior〉]+
      [〈Orchestrator_Constraint〉]+
End Orchestrator
```

编排端口(Port):编排端口是编排者与外部环境的交互

点，其定义方式与前面基本服务端口的定义方式相同。

编排结构(Orchestrator_Structure)：编排结构描述了参与编排的多个参与者(Participant)之间的拓扑关系，由一个连接件和若干个参与者组成。可以用一个体系结构配置描述。内部结构定义如下：

```
〈Orchestrator_Structure〉::= Configure:〈NormalConfigure〉
/指向一个描述内部结构的体系结构配置
End Orchestrator_Structure
```

编排行为(Orchestrator_Behaviour)：描述了编排者如何协同多个参与者服务的组合过程，编排行为表现为多个服务的结构化组合行为。其定义如下：

```
〈Orchestrator_Behaviour〉::=
Orchestrator _Behaviour
        [Service_Behaviour]+
End Orchestrator_Behaviour
```

编排约束(Orchestrator_constraint)：描述了编排者内部结构和编排过程必须遵守的限制。其定义方式同样采用一阶谓词逻辑的断言来表示。

4.3 常规连接件

SOADL-EH 语言中显式地定义连接件为第一类实体，用来描述服务和编排者之间的连接和交互关系。每个常规连接件分为接口和交互协议两部分。接口由一组角色(role)定义，表明参与编排的参与者所应有的外部行为。交互协议描述如何将角色粘连(Glue)在一起产生交互，从而将参与交互的参与者计算规范组合起来。其定义如下：

```
〈NormalConnector〉::=NormalConnector〈NCName〉
  [〈NormaConnector_Role〉]+
  [〈NormalConnector_Behavior〉]+
End NormalConnector
```

常规连接角色(NormalConnector_Role)：规定了连接在常规连接件上的交互方的单个接口的操作和消息。常规连接的操作和消息定义方式与前面在基本服务定义的方式相同。其角色定义如下：

```
〈NormalConnector_Role〉::=Role〈NCRoleName〉
        [〈NC_Role_Type〉]+
        [〈MessageTypes〉]+
        [〈Operation〉]+
End NormalConnector_Role
〈NC_Role_Type〉::= provider | requester
```

常规连接行为(NormalConnector_Behavior)：刻画了连接件角色之间的交互规范。常规连接行为由一组原子连接动作(ConnectAction)组成。其定义如下：

```
〈NormalConnector_Behaviour〉::=
NormalConnector_Behaviour〈NCName〉
        [〈ConnectAction〉]+
End NormalConnector_Behaviour
```

连接动作(ConnectAction)定义同样可以采用文献[15]中基于 π 演算类似的方式来描述。

4.4 异常处理服务

异常处理服务(EHService)描述对异常源所引发异常的处理。异常处理服务包括对外提供的异常处理功能接口、支持异常处理的内部结构、设计约束和实现异常处理的过程描述。其定义如下：

```
〈EHService〉::=EHService〈EHService_Name〉
        [〈Port〉]+
        [〈EHService_Structure〉]
      [〈EHService_Behaviour〉]+
      [〈EHService_Constraint〉]+
End EHService
```

异常处理服务端口(Port)是异常处理服务与外部环境的交互点。其定义方式与前面基本服务端口的定义方式相同，定义如下：

```
〈Port〉::= Port〈port_name〉
        [〈Port_role〉]+
        [〈MessageTypes〉]+
        [〈ExceptionTypes〉+]
        [〈Operation〉]+
End Port
```

异常处理服务结构(EHService_Structure)：异常处理服务有 0 个或 1 个内部结构，异常处理服务内部结构刻画了参与异常处理的参与者之间的拓扑关系；异常处理服务的内部结构由一个异常连接件和若干个异常处理服务组成。可以用一个体系结构异常配置描述。其定义如下：

```
〈EHService_Structure〉::=
  EHService_Structure〈name〉
        EHConfigure:〈EHConfigure〉
//指向一个描述异常处理内部结构的异常配置
End EHService_Structure
```

异常处理服务行为(EHService_Behaviour)描述了异常处理服务如何处理异常的过程。异常处理服务约束(EHService_Constraint)规定了异常处理服务所要遵守的设计限制；异常处理服务的行为和约束定义与编排者中的行为和约束定义相同。

4.5 异常连接件

异常连接件(EHConnector)是 SOADL-EH 中表示异常捕获和转发的语言设施。异常连接件描述异常源和异常处理服务之间的连接和交互协议。每个异常连接件至少包括一个异常接口和交互协议(EHGlue)。接口由一组角色(EHrole)定义，表明参与异常处理逻辑的组件的外部行为；交互协议由一组异常处理策略(EHPolicy)组成，每条策略描述从异常源(ESource)捕获异常、转发异常到适当的异常处理器(EHandler)以及异常处理回到返回点(EHReturnPoint)的规则。其定义如下：

```
〈EHConnector〉::=EHConnector〈EHName〉
      [〈EHrole〉]+
      [〈EHGlue〉]+
End EHConnector
```

异常连接件角色(EHrole)：异常连接件角色规定了挂接在异常连接件上的交互方的每个接口的消息、操作和行为。异常连接件角色的消息和操作定义方式与前面在常规连接件中定义方式相同。其定义如下：

```
〈EHrole〉::=EHrole〈role_Name〉
        [〈EHrole_type〉]
        [〈messageTypes〉]+
        [〈operation〉]+
End EHrole
〈EHrole_type〉::= provider | requester
```

异常连接件的交互协议(EHGlue)通过一组异常处理策

略(EHPolicy)来描述异常源(ESource)、异常处理器(EHHandler)、异常处理返回点(EHReturnPoint)之间的交互关系。其定义如下：

〈EHPolicy〉::=EHPolicy〈EHPolicy_Name〉
[〈ESource〉]+
[〈EHandler〉]+
[〈EHReturnPoint〉]
End EHPolicy

异常源(ESource)包括正常处理逻辑中的基本服务(Service)和编排者(Orchestrator)以及异常处理服务(EHService)。其定义如下：

〈ESource〉::= Service | Orchestrator | EHService

异常处理器(EHandler)由一组异常处理服务(EHService)组成。其定义如下：

〈EHandler〉::=[EHService]+

异常处理返回点(EHReturnPoint)描述异常处理器处理异常后的返回信息。异常处理器对异常的处理会产生不同的结果和影响。①异常处理器成功完成异常处理，则返回到异常源所在位置；②异常处理器无法成功处理异常，则抛出(Throw)接收到的异常并返回到异常源位置；③异常处理器处理异常时又引发(Raise)新的异常，这种情况下异常处理器抛出新的异常后回到返回点。异常处理返回点定义如下：

〈EHReturnPoint〉::=Service| Orchestrator | EHService

4.6 配置

体系结构配置(Configuration)描述了系统的总体拓扑结构。SOADL-EH对系统体系结构配置的描述分为常规配置和异常配置两部分。

〈Configuration〉::=Configuration
[〈NormalConfigure〉]+
[〈EHConfigure〉]+
End Configuration

4.6.1 常规配置

体系结构常规配置(NormalConfigure)描述了软件系统正常业务逻辑的拓扑结构。包括基本服务实例定义、编排者实例定义、常规连接件实例定义、基本服务实例的端口与常规连接件实例的角色之间的绑定以及编排者实例的端口与常规连接件实例的角色之间的绑定。其定义如下：

〈NormalConfigure〉::=NormalConfigure〈c_name〉
[〈Service_Instance_Def〉]+
[〈Orchestrator_Instance_Def〉]+
[〈NormalConnector_Instance_Def〉]+
[〈Service_Role_Binding〉]+
[〈Orchestrator_Role_Bingding〉]+
End NormalConfigure

其中，

〈Service_Instance_Def〉::=
〈Service_Instance_Name〉:〈Service_Name〉
〈Orchestrator_Instance_Def〉::=
〈Orchestrator_Instance_Name〉:
〈Orchestrator_Name〉
〈NormalConnector_Def〉::=
〈NormalConnector_Instance_Name〉
〈NormanlConnector_Name〉
〈Service_Role_Binding_Def〉::=
〈Service_Instance_Name〉.〈Port_Name〉 as
〈NormalConnector_Instance_Name〉.〈Role_Name〉
〈Orchestrator_Role_Binding_Def〉::=
〈Orchestrator_Instance_Name〉.〈Port_Name〉 as
〈NormalConnector_Instance_Name〉.〈Role_Name〉

4.6.2 异常配置

体系结构异常配置(EHConfigure)描述了软件系统异常处理逻辑的拓扑结构。包括基本服务实例定义、编排者实例定义、异常处理服务定义、异常连接件实例定义、基本服务实例的端口与异常连接件实例的角色之间的绑定、编排者实例的端口与异常连接件实例的角色之间的绑定和异常处理服务的端口与异常连接件实例的角色之间的绑定。其定义如下：

〈EHConfigure〉::=EHConfigure〈c_name〉
[〈Service_Instance_Def〉]+
[〈Orchestrator_Instance_Def〉]+
[〈EHService_Instance_Def〉]+
[〈EHConnector_Instance_Def〉]+
[<Service_Role_Binding_Def〉]+
[〈Orchestrator_Role_Bingding_Def〉]+
[〈EHServie_Role_Binding_Def〉]+
End EHConfigure

其中，

〈Service_Instance_Def〉::=
〈Service_Instance_Name〉:〈Service_Name〉
〈Orchestrator_Instance_Def〉::=
〈Orchestrator_Instance_Name〉:
〈Orchestrator_Name〉〈EHService_Instance_Def〉::=
〈EHService_Instance_Name〉:〈EHService_Name〉〈EHConnector_Instance〉::=
〈EHConnector_Instance_Name〉:
〈EHConnector_Name〉:
〈Service_Role_Binding_Def〉::=
〈Service_Instance_Name〉.〈Port_Name〉 as
〈EHConnector_Instance_Name〉.〈Role_Name〉
〈Orchestrator_Role_Binding_Def〉::=
〈Orchestrator_Instance_Name〉.〈Port_Name〉 as 〈EHConnector_Instance_Name〉.〈Role_Name〉
〈EHService_Role_Binding_Def〉::=
〈EHService_Instance_Name〉.〈Port_Name〉 as 〈EHConnector_Instance_Name〉.〈Role_Name〉

结束语 本文针对构建具有容错能力的面向服务软件体系结构的科学问题，提出了一种新型的支持异常处理的面向服务体系结构描述语言SOADL-EH。该语言不但具备当前ADL描述面向服务软件体系结构的能力，通过扩展异常、异常处理服务、异常连接件及异常处理等体系结构语言成分，将体系结构元素划分为处理正常行为和异常行为两部分，还能够清晰地表示和规约基于异常处理的具有容错能力的面向服务软件体系结构。

未来的研究工作首先是要加强SOADL-EH的语义描述能力，引入形式化方法对基本服务、编排者、常规连接件、异常处理服务、异常连接件及配置等体系结构元素进行精确的描述，从而能够进一步增强在语义层次上对系统行为进行高层分析和推理的能力。其次是SOADL-EH的自动化支撑工具的开发，开发可视化建模和分析工具，支持异常处理的面向服务软件体系结构的表示和形式化分析。

参 考 文 献

[1] 邢少敏,周伯生. SOA 研究进展[J]. 计算机科学,2008,35(9):13-20

[2] Papazoglou M P, Traverso P, Dustdar S, et al. Service-Oriented Computing: State of the Art and Research Challenges[J]. IEEE Computer, 2007, 40(11): 38-45

[3] Tartanoglu F, Issarny V, Levy N, et al. Dependability in the Web service architecture[C] // Architecting Dependable Systems, LNCS 2677. SpringerVerlag, 2003: 89-108

[4] Medividovic N, Taylor RN. A classification and comparison framework for software architecture description language[J]. IEEE Transactions on Software Engineering, 2000, 26(1): 77-93

[5] W3C. Web Services Architecture [EB/OL]. http://www.w3.org/TR/ws-arch/, 2004

[6] Miladi M N, Krichen I, Jmaiel M, et al. An xADL extension for management dynamic deployment in distributed service oriented architectures[C] // Proceedings of the Third IPM International Conference on Fundamentals of Software Engineering. Lecture Notes in Computer Science, vol. 5961. Berlin: Spring, 2009

[7] Sanz M L, Qayyum Z, et al. Representing service-oriented architectural models using pi-adl[C] // ECSA 2008, LNCS 5292. 2008: 273-280

[8] 杨鑫,陈俊亮. WSC/ADL: Web Service 组合系统体系结构描述语言[J]. 软件学报,2006,17(5):1182-1194

[9] 饶元,冯博琴,李尊朝. ALBC4WS:一种基于软件体系结构生命周期的动态服务合成框架[J]. 计算机研究与发展,2005,42(12):2063-2069

[10] Jia Xiang-yang, Ying Shi, et al. A New Architecture Description Language for Service-Oriented Architecture[C] // Proceeding of Sixth International Conference on Grid and Cooperative Computing(GCC 2007). Aug. 2007: 96-103

[11] 蒋哲远,韩江洪,王釗. 面向服务软件体系结构的 XML 描述和构造[J]. 小型微型计算机系统,2008,29(8):1437-1444

[12] Castor F F, Brito P H, Rubira C M. Specification of exception flow in software architecture[J]. The Journal of System & Software, 2006, 79(10): 1397-1418

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

[14] de G Castro P A, Rubira C, de Lemos R. A Fault-Tolerant Software Architecture for Component-Based Systems[C] // Architecting Dependable Systems, LNCS 2677. Spring, 2003: 129-149

[15] de Lemos R, de Castro P A G, Rubira C M. A faulttolerant architectural approach for dependable systems[J]. IEEE Software, 2006, 23(2): 80-87

[16] Oquendo F. π-ADL: An Architecture Description Language based on the Higher-Order Typed π-Calculus for Specifying Dynamic and Mobile software Architectures[J]. Software Engineering notes, 2004, 29(3): 1-14

(上接第 121 页)

结束语 Web 服务 QoS 的准确性是评价和选择服务的重要依据。为了提高 Web 服务 QoS 的准确度,本文设计了一种基于事例推理的 QoS 动态预测方法,首次将引起 Web 服务 QoS 波动的主要因素与服务的 QoS 关联起来,依据事例推理技术动态地预测 Web 服务处理具体任务时的 QoS,弥补了已有的 QoS 度量、预测方法只挖掘历史 QoS 数据的规律性而不考虑 Web 服务所在的环境、任务类型、任务大小对 Web 服务 QoS 的影响这一不足。经过试验证明,本预测方法能够大幅度提高 QoS 的准确性。但本文中的预测方法是即时预测,即一旦有任务请求就立即开始预测,不能处理在未来某一时刻点开始调用服务的 QoS 预测问题,这将是我们以后研究的重点。

参 考 文 献

[1] Vu L H, Hauswirth M, Aberer K. QoS-based Service Selection and Ranking with Trust and Reputation Management [C] // Proceedings of the International Conference on Cooperating Information System(CoopIS 2005). Agia Napa, Cyprus, 2005: 466-483

[2] Zeng L Z, Benatallah B, Ngu A H H, et al. QoS-aware middleware for Web services composition [J]. IEEE Trans. on Software Engineering, 2004, 30(5): 311-327

[3] 邵凌霜,周立,赵俊峰,等. 一种 WebService 的服务质量预测方法[J]. 软件学报,2009,20(8):2062-2073

[4] Zheng Zibin, Ma Hao, Lyu M R, et al. WSRec: A Collaborative Filtering Based Web Service Recommender System [C] // IEEE International Conference on Web services. 2009: 437-444

[5] 黄景文,胡志华. Web 服务 QoS 的免疫多信号预测模型研究[J]. 广西大学学报:自然科学版,2009,34:535-539

[6] 刘克非,王红,许作萍. 一种基于服务质量预测的 Web 服务选择方法[J]. 计算机技术与发展,2007,17:103-105

[7] Gao Zheng-dong, Wu Geng-feng. Combing QoS-based Service Selection with Performance Prediction [C] // Proceedings of the 2005 IEEE International Conference on e-Business Engineering (ICEBE'05)

[8] Li Mu, Huai Jin-peng, Guo Hui-peng. An Adaptive Web Service Selection Method Based on the QoS Prediction Mechanism [C] // IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology-Workshops. 2009

[9] Hwang San-yih, Wang Hao-jun, Tang Jian, et al. A probabilistic approach to modeling and estimating the QoS of Web service-based workflows[J]. Information Sciences, 2007, 177: 5484-5503

[10] Schank R. Dynamic Memory: A Theory of Learning in Computers and People [M]. Cambridge University Press, 1982

[11] Li Hui, Sun Jie, Sun Bo-liang. Financial distress prediction based on OR-CBR in the principle of k-nearest neighbors [J]. Expert System with Applications, 2009, 36: 643-659

[12] Varma A, Roddy N. ICARUS: design and development of a case-based reasoning system for locomotive diagnostics [J]. Engineering. Applications of Artificial Intelligence, 1999, 12(6): 681-690

[13] 屈利,苑津莎,李丽. 基于事例推理的电力系统短期负荷预测[J]. 电力科学与工程,2008,24(2):59-63

[14] Sankarkp S. Foundations of soft case based reasoning [M]. New York: Wiley, 2003

[15] Walkerden F, Jeffery D R. An empirical study of analogy-based software effort estimation [J]. Empirical Software Engineering, 1999, 4: 135-158

[16] Shepperd M, Schofield C. Estimating software project effort using analogies [J]. IEEE Transactions on Software Engineering, 1997, 23: 736-743

[17] Kadoda G, Cartwright M, Chen L, et al. Experiences using case based reasoning to predict software project effort [C] // Proceedings EASE 200 Conferences. Keele, UK, 2000