ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.17, No.6, June 2006, pp.1318−1327
DOI: 10.1360/jos171318

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

建模样式:一种评估软件体系结构非功能属性的方法*

徐 鹏+, 杨放春

(北京邮电大学 计算机科学与技术学院,北京 100876)

Modeling Patterns: A Method to Evaluate Non-Functional Attributes of Software Architectures

XU Peng+, YANG Fang-Chun

(School of Computer Science and Technology, Beijing University of Posts and Telecommunications, Beijing 100876, China)

+ Corresponding author: Phn: +86-10-62251188, Fax: +86-10-62241658, E-mail: xupengpeng@263.net, http://www.bupt.edu.cn

Xu P, Yang FC. Modeling patterns: A method to evaluate non-functional attributes of software architectures. *Journal of Software*, 2006,17(6):1318−1327. http://www.jos.org.cn/1000-9825/17/1318.htm

Abstract: The design of software architecture plays an important role in a software process. In the design phase, the non-functional feature evaluation of software architecture would contribute a lot in providing high quality software products. Modeling patterns, which is an extension of UML (unified modeling language), are brought forward for software non-functional feature evaluation in this paper. Furthermore, an example, in which modeling pattern "AvailabilityChain" is used to evaluate the availability of software architecture "1 Message Distributor—*n* Message Processors", is given to illustrate the application of modeling patterns. Moreover, to support the application of modeling patterns, "Modeling Pattern Knowledge-Base", in which modeling patterns and reference values of tags used by each modeling pattern are managed, is involved. UML-based Patterns and Modeling Pattern Knowledge base will simplify the evaluation of software architecture in the software process and contribute a lot in providing high quality software products.

Key words: software architecture; unified modeling language; non-functional; modeling; pattern; availability; knowledge-base

摘 要: 软件体系结构设计是软件过程中最为重要的环节之一.在设计阶段完成对软件体系结构非功能属性的评估,对于高质量软件产品的开发非常重要.通过对统一建模语言(UML)的扩展,提出了"建模样式"用于在软件设计阶段对软件体系结构非功能属性进行评估,并结合"可用性链"建模样式在分析软件体系结构"单消息分发-多消息处理"可用性中的应用,给出了建模样式的使用示例.同时,针对建模样式的应用,还提出了"建模知识库"用于管理和维护建模样式,提供各建模样式中标签的参考值.基于 UML 的建模样式以及建模知识库的使用,可以简化对软件体系结构非功能属性评估的复杂度和工作量,使其可以为软件开发人员所用,并融入到高质量软件开发过程中.

关键词: 软件体系结构;统一建模语言;非功能;建模;样式;可用性;知识库

* Supported by the Program for Changjiang Scholars and Innovative Research Team in University/ PCSIRT (长江学者和创新团队发展计划)

Received 2006-01-09; Accepted 2006-03-28

中图法分类号: TP311 文献标识码: A

软件体系结构是描述软件单元(element)、软件单元的属性(property)以及这些单元之间关系(relationship)的结构[1].因此,软件体系结构直接决定了软件系统的运行框架,其优劣不但决定了软件系统是否可以满足针对此软件的功能需求,而且还决定了这些功能需求是否能被合理、高效地实现,即也决定了软件系统基本的非功能属性[2].这里,非功能属性不但包括时域性能,而且还包括诸如可用性、可扩展性、可维护性等一系列属性.

在当前的软件工程研究中,对于软件体系结构非功能属性的评估已经有了一系列研究.目前,评估软件体系结构非功能属性主要可以采用两类技术:提问(questioning)和测量(measuring)[3].提问技术中将针对软件体系结构提出一系列定性分析的问题,并通过对这些问题的解答完成对软件体系结构的某种非功能属性的评估;而测量技术中则通过对软件体系结构进行一系列定量测量,完成对特定非功能属性的评估.一般来说,提问技术比测量技术在对各种非功能属性的适应能力方面有一定的优势;而测量方式在评估的精确程度上要优于提问方式.因此在具体应用中,往往需要综合运用“提问”和“测量”两种方式对软件体系结构的非功能属性进行评估[4].这里需要特别指出的是:在具体应用中,不论采用“提问”还是“测量”,都将软件体系结构非功能属性评估作为一个独立的过程,并没有将这个过程与软件开发过程结合起来.

另一方面,对于在软件开发过程中对软件体系结构非功能属性进行评估来说,更为重要的是,评估软件体系结构非功能属性的各种方法是否可以被软件工程师掌握,并应用于具体的软件系统开发中[5].因此,使用易于为软件开发人员所用的技术和方式,对于将软件体系结构非功能属性评估切实应用于实践非常重要.

本文针对上述问题,通过对统一建模语言(unified modeling language,简称 UML)的扩展提出了建模样式(modeling pattern).建模样式可以与软件开发过程中使用的 UML 相结合,直接使用软件设计中产生的 UML 模型生成相应的非功能属性评估模型,从而使软件开发人员可以在完成软件功能属性设计的同时,完成对软件设计的非功能属性的评估.

本文第 1 节对建模样式加以说明.在第 2 节中以“可用性链”建模样式在分析软件体系结构“单消息分发-多消息处理”可用性中的应用对建模样式的应用方式加以介绍.第 3 节提出“建模知识库”,以支持建模样式在软件体系结构设计中的实际应用.

1 建模样式技术

建模样式是样式(pattern)技术在软件工程领域的一个具体应用.建模样式包括一系列将方案抽象为系统模型(数学模型和/或仿真模型)的符号、方法,一系列与特定系统模型相应的数学理论和仿真方法及其适用范围.软件设计人员可以套用这些方法迅速建立软件非功能属性模型并加以分析,从而缩短系统建模、分析过程的时间,降低工作难度.同时,本文中提出的建模样式没有引入新的形式化描述手段,而是基于 UML 完成对建模样式的形式化描述,从而极大地增强了本文提出的建模样式的实用性.

与“样式”的一般概念相同,建模样式也涉及 4 个基本要素:名称、问题、解决方案、效果[6].

1. 名称(name):名称是样式的标识,是一个助记名,用于描述样式针对的问题、解决方案和效果.

2. 问题(problem):描述了此样式的使用范围,解释了建模问题和问题存在的前因后果,往往还指明了此样式的应用场景.

3. 解决方案(solution):描述了具体的建模方法,涉及如何从 UML 图中导出非功能属性模型、指明模型中待定义的参数、如何分析此模型等多方面的内容.因为样式就像一个模板,可以应用于多种不同场合,所以,解决方案并不描述一个特定而具体的模型的设计和实现,而是提供对通用方法的抽象描述.本文还进一步提出了建模样式解决方案的框架.这个框架主要包括 4 个组成部分:

1) 带标签(tag)的 UML 对象(UML object with tag,简称 UOT).UML 对象包括 UML 模型中涉及的各种元素,如关联(association)、对象(object)等.

2) 标签完备性检查规则(tag integrity rules,简称 TIR).此规则说明了保证建模所需数据完备性和正确性的

规则.

3) 模型(model).由 UML 图导出的模型是建模分析的基础.本文并未规定“模型”的描述方式,可以采用诸如图形、自然语言等各种方式.模型的存在主要是为了使开发人员或是系统分析人员对模型有直观的认识.

4) 数学模型(mathematic model).数学模型是建模分析的理论依据.正是数学模型连接了模型和分析结论.

4. 效果(consequences):描述了样式应用的效果以及使用样式应权衡的问题.对于建模样式主要是指应用特定样式进行系统建模时可能出现哪些误差、误差会导致哪些分析的不准确、应作何种补偿等.

1.1 建模样式的UML描述

UML 得以广泛应用的一个非常重要的原因就是 UML 可扩展.UML 提供的扩展机制包括[7]:

1) 构造型:允许构造新的构造块.此构造块既可以从现有构造块派生,又专门针对被研究的问题;

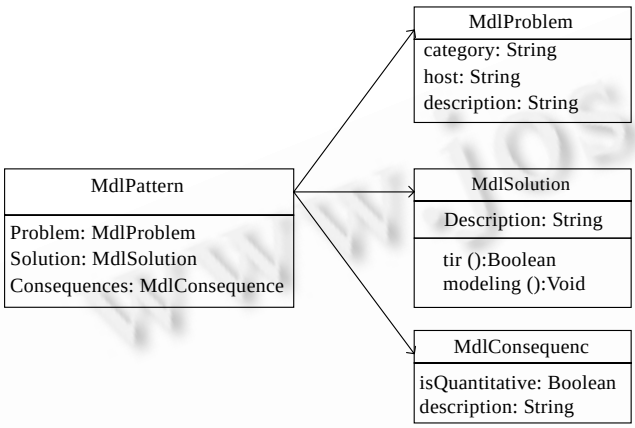

Fig.1 Class diagram of modeling pattern

图 1 建模样式类图

2) 标签(tag):扩展了 UML 构造块的特性,允许创建详述元素的新信息;

3) 约束:扩展了 UML 构造块的语义,允许增加新的规则或修改现有的规则.

UML 中规范了针对上述扩展机制的图形和文字的表示方法.UML 用户可以根据自身的需要灵活地对 UML 加以扩展以适应特定领域.本文中主要采用 UML 提供的标签机制对 UML 进行扩展,使之可以被用于描述建模样式.更为重要的是,基于标签机制的扩展,可以保证扩展后的 UML 模型仍能被软件开发人员理解并掌握,从而可以被无缝地运用于软件开发中.“建模样式”采用如图 1 所示数据结构描述.

图 1 中各类的属性和方法说明如下:

MdlProblem

• MdlProblem.category:此属性说明该建模样式针对的非功能属性,取值如 TIME_PERFORMANCE(时域性能,如时延等),AVAILABILITY(可用性)等;

• MdlProblem.host:此属性说明该建模样式将对 UML 中某一种或是几种图(diagram)进行建模,取值如 USE_CASE(用例图(use case diagram)),INTERACTION(交互图(interaction diagram),COLLABORATION(协作图(collaboration diagram))等.由于在 UML 中描述软件体系结构往往更多地使用用例图、合作图和交互图,所以在本文介绍的针对软件体系结构的建模样式中,host 主要是用例图、交互图和合作图;

• MdlProblem.description:此属性对该建模样式针对的问题加以进一步阐述,一般是一段描述语言,如 Z 语言等.考虑到建模样式针对的应用场景的多样性,这里也可以使用自然语言.

MdlSolution

• MdlSolution.model():此方法提供了将特定 UML 图转化为“模型(model)”的机制(算法).这里所谓“算法”将要包括如下内容:

  ◆ 此建模样式的使用中需要为哪些 UML 对象“粘贴”标签(tag)及各标签的类型;

  ◆ 如何利用 UML 对象以及标签提供的数据得出 MdlProblem.category 相关的指标.

• MdlSolution.tir():此方法用于检查建模所需数据的完备性和正确性.

MdlConsequence

• MdlConsequence.isQuantitative:此属性说明该建模样式是完成定性分析,还是完成定量分析.因为在 MdlConsequence 中没有属性用于说明分析的结论,所以在特定的建模样式中,往往使用 MdlConsequence 的子类.该子类将在 MdlConsequence 的基础上增加特定的属性以说明建模的结论.这里需要特别指出的是,在分析

软件体系结构的非功能属性时往往不能、也不需要给出非常精确的定量分析结果,只要做到“足够精确、够用于指导进一步的软件设计”就可以了.

Tag

标签是建模样式中的一个非常重要的概念.关联于特定 UML 对象的标签指明了建模的输入数据;关联于模型的标签指明了模型的各种参数.考虑到数据类型的多样性,Tag 可以进一步派生为 StringTag,IntegerTag 和 FloatTag;针对不同的建模样式往往还需要将这些标签加以组合,从而派生出更高层的标签,如图 2 所示.

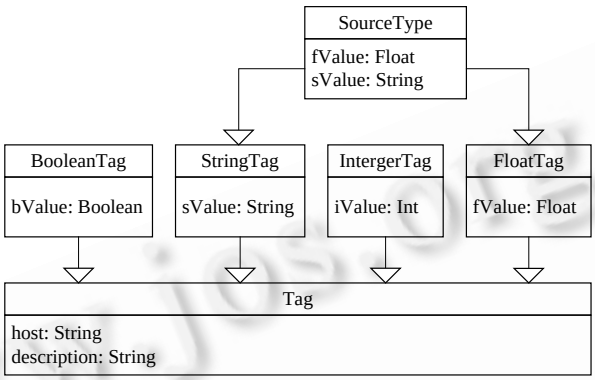

Fig.2 Class diagram of tag (inheritance relationship)

图 2 标签类图(继承关系)

- Tag.host:此属性说明此标签可以被关联至 UML 中某一种或是几种对象(如 ACTOR(活动者(actor), ASSOCIATION (关联(association)等),或是被关联至由建模样式引入的模型(如源(source)、队列(queue)等).

由于 MdlPattern 仅提供了建模样式的框架.具体建模样式将作为子类继承 MdlPattern.建模样式(MdlPattern)的子类(如 QueuingNetworks, AvailabilityChain)可以进一步被继承(如 MMnmQueuingNetwork),从而进一步细化建模样式的应用.如图 3 所示.

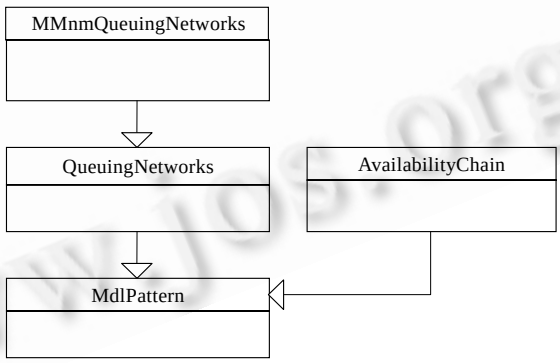

Fig.3 Inheritance relationship of MdlPattern

图 3 建模样式的继承关系

1.2 建模样式示例:**AvailabilityChain**(可用性链)

“可用性链”是指在系统中完成特定消息(函数调用也可以被看作是消息)处理所需的一系列模块、子系统、组件、构件等连接而成的一条前后相继的“链”.每条“链”都与某一个或是某一类消息对应.“可用性链”主要由两类元素构成:完成消息处理的节点(nodes)和用于传送消息的连接(connectors).当且仅当一条“可用性链”上所有元素都可用时,此“链”可用.基于“可用性链”的概念,整个系统的可用性可以被分解为多条同时存在的“可用性链”的可用性.只有当所有“可用性链”都可用时,系统才可用.另外,由于系统中各类消息出现的概率和重要性不

同,各个(类)消息相应的“可用性链”的权值也应有所不同.基于上述分析,“AvailabilityChain(可用性链)”建模样式可以表示如下:

• AvailabilityChain.problem.category=“AVAILABILITY”

AvailabilityChain 样式主要被用于对软件体系结构的可用性进行建模分析.

• AvailabilityChain.problem.host∈{“USE_CASE”,“INTERACTION”,“COLLABORATION”,“OTHER”}

AvailabilityChain 样式适用于 UML 中规范的用例图、交互图、协作图等.OTHER 是指其他没有明确列出的 UML 要素,以提供必要的扩展能力.

AvailabilityChain.solution 是 MdlSolution 的派生类,如图 4 所示.

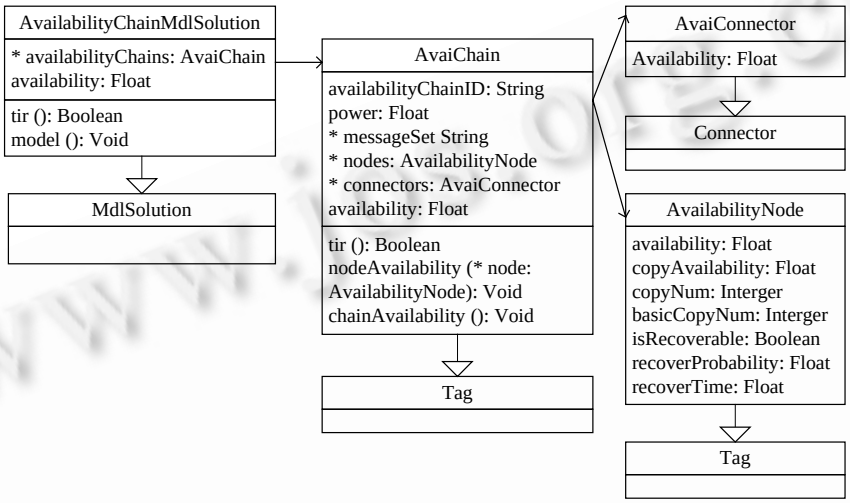

Fig.4 Class diagram of AvailabilityChainMdlSolution

图 4 可用性链建模样式解决方案类图

AvaiChain 类是 AvailabilityChain 样式中最为重要的类.此类规范了“可用性链”的属性和方法.

• AvaiChain.availabilityChainID:String 给出本“可用性链”唯一的标识.

• AvaiChain.power:Float 指明此“可用性链”的权值.“可用性链”的权值将随着此“链”相应的消息集的重要性的提高而增大,但应小于等于 1.

• AvaiChain.(* messageSet):String,此字符串数组列出了与此可用性链相应的所有的消息.

• AvaiChain.(* nodes):AvailabilityNode 指明了此“链”中所有参与消息处理的节点.

AvaiChain.(* nodes).availability:Float 给出此节点的可用性;AvaiChain.(* nodes).copyAvailability:Float 给出此节点的副本的可用性.

AvaiChain.(* nodes).copyNum: Interger 指明此节点的处理能力的副本个数.当此节点中处理能力的一个副本失效后,其他副本可以接替此副本的工作.

AvaiChain.(* nodes).basicCopyNum:Interger 指明此节点的处理能力副本个数的最小值.即当此节点中处理能力的副本数不小于此数值时,本节点内其他副本可以接替失效副本的工作,从而保证整个节点的可用性.

AvaiChain.(* nodes).isRecoverable:Boolean 指明此节点在发生失效后是否可以恢复.

AvaiChain.(* nodes).recoverProbability:Float 指明此节点在发生失效后可以恢复的概率.

AvaiChain.(* nodes).recoverTime:Float 指明此节点在发生失效后到恢复运行的时间间隔.

AvaiChain.(* nodes).host:String∈{“CLASS”,“COMPONENT”, “OTHER”},指明“AvaiChain”为针对“类”、“组件”或是“其他(OTHER)”的标签;OTHER 是指其他没有明确列出的 UML 要素,以提供必要的扩展能力.

• AvaiChain.(* connectors):AvaiConnector 指明了此“链”中所有用于消息传送的“连接”.

AvaiChain.(* connectors).availability:Float 指明该“连接”的可用性.

AvaiConnector 是 Connector 类的派生类.因此(* connectors)也继承了 Connector 类的各个属性.

* connectors.end1:String 指明此连接的一侧的端点.

* connectors.end2:String 指明此连接的另一侧的端点.

* connectors.head:String 指明此连接的头侧的端点.此连接的方向为从另一个端点到头端点.当 head 指向一个空字符串时,此连接无方向性.

* connectors.host:String={“INTERACTION”},指明“connector”为针对 INTERACTION 的标签.

• AvaiChain.Availability:Float 指明了此“可用性链”的可用性.

• AvaiChain.tir():Void 主要完成对 AvaiChain 类中的各个属性(数据)的完备性的检查.在此完备性检查中非常重要的一点就是检查“可用性链”中的所有“连接”相应的节点是否都被包含在“可用性链”中.

• AvaiChain.nodeAvailability(* node: availabilityNode):Void 完成对 node 指向的节点的可用性计算.此计算遵循如下数学模型:

假定模块 $M_1,M_2,\ldots,M_n$ 都可以完成相同的处理,其可用性为 p,且当 n 个模块中 k+1 个模块发生故障时系统才会失效,即系统的冗余度为 $r=\frac{k}{n-k}$,则一个需要由上述模块并行处理的操作的可用性为

$$R_p=\sum_{i=n-k}^{n}p^i(1-p)^{n-i}\ .$$

将 p=node->copyAvailability,n=node->copyNum,k=node->copyNum−node->basicCopyNum,R_p=node->availability 代入上述公式即为节点可用性计算公式.AvaiChain.nodeAvailability( )就是此公式的函数实现.

以上代码说明节点的可用性(node->availability)是节点中各个副本的可用性(node->copyAvailability)的 node->copyNum 次方.

• AvailabilityChains:AvaiChain 列出了系统中所有的“可用性链”,表现在特定 UML 图上就是一系列有向虚线.

• availability:Float 指明此系统的可用性.

• tir()完成对 AvailabilityChain 中的各个属性(数据)的完备性的检查,其中非常重要的一个检查项就是确定各可用性链的权值之和是否为 1.如果不等于 1,则说明某节点权值的附值有误.

• model()完成对整个系统可用性的计算.整个系统的可用性将是系统中标注的所有“可用性链”的可用性之积.

2 建模样式在分析软件体系结构非功能属性中的应用

2.1 “单消息分发-多消息处理”软件体系结构

“单消息分发-多消息处理(1 MD-*n* MP)”软件体系结构是在“支持多媒体和移动业务的呼叫控制软件系统”的研发中提出的一种软件体系结构.这种软件体系结构借鉴了“代理(broker)”软件体系结构的设计思想[6].“Broker”是消息处理软件系统中最为常用的一种软件体系结构样式.虽然“Broker”可以有效地解决系统功能分布的问题,但是这种结构为解决远程调用而引入的多个实体(主要是各种 proxy 和 broker)导致此软件体系结构较为复杂.为此,在“1 MD-*n* MP”中,将 Broker 中的各种 proxy 和 broker 合并为统一的消息分发器(message distributor,简称 MD),保留了消息处理器(message processor,简称 MP).其中,MD 作为模块与外界的统一接口,一方面从其他模块接收消息,将消息转发至 MP;另一方面从 MP 接收消息,将消息发送到其他模块.同时,由于一般情况下消息分发的处理工作较为简单,所以多个消息处理器可以共用同一消息分发器. “1 MD-*n* MP”软件体系结构的对象图如图 5 所示.

MsgDistributor 的各个方法的说明如下:

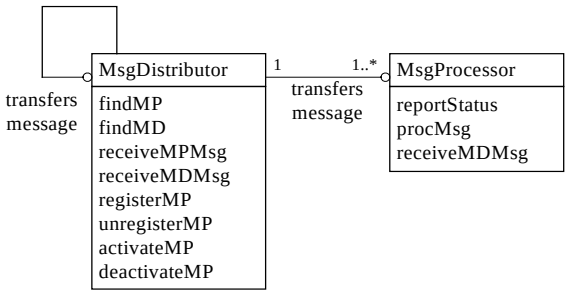

Fig.5 Class diagram of software architecture "1 MD-*n* MP"

图 5 软件体系结构"1 MD-*n* MP"的类图

Msg.Distributor.unregisterMP,用于将一个 MP 从 MD 中注销,需要指出的是,在一个 MP 被注销之前,此 MP 应被从 MD 中去激活.

Msg.Distributor.activateMP,用于将一个已完成注册的 MP 激活.在完成激活后,MD 将可以把消息发送至此 MP 进行处理.

Msg.Distributor.deactivateMP,用于将一个已完成注册的 MP 去激活.在去激活后,MD 将不能再把消息发送至此 MP 进行处理.

MsgProcessor 的各个方法的说明如下:

MsgProcessor.reportStatus,用于给出本 MP 的当前状态.当 MD 向 MP 发送消息时,可能会使用此状态信息.

MsgProcessor.procMsg,用于完成对收到的消息的处理.针对具体的应用应重载此方法.

MsgProcessor.receiveMDMsg,用于从 MD 处接收消息.

Msg.Distributor.findMP,用于从多个处于"激活"状态的 MP 中选择一个处理当前消息.

Msg.Distributor.findMD,选择接收待发送消息的 MD.

Msg.Distributor.receiveMPMsg,用于从 MP 处接收消息.Msg.Distributor.receiveMDMsg,用于从 MD 处接收消息.

Msg.Distributor.registerMP,用于将一个 MP 注册至 MD,需要指出的是,在一个 MP 注册至 MD 之后,MD 还不会将消息发送给此 MP 处理,直至此 MP 被激活.

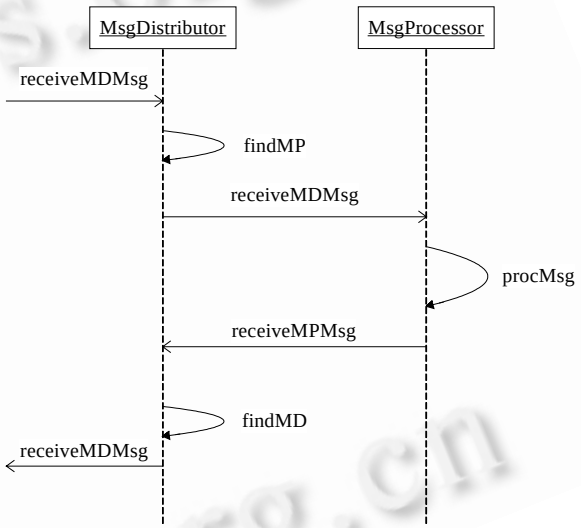

Fig.6 Message sequence chart of "1 MD-*n* MP"

图 6 "1 MD-*n* MP"的消息序列图

"1 MRD-*n* MP"软件体系结构完成消息处理的消息序列图如图 6 所示.

2.2 "AvailabilityChain"建模样式的应用

"1 MRD-*n* MP"软件体系结构的设计更多地考虑了"支持多媒体和移动业务的呼叫控制软件系统"的功能需求.在完成了针对功能需求的软件体系结构设计后,采用建模样式对软件体系结构的非功能属性进行评估.本文将基于 AvailabilityChain 对"单消息分发——多消息处理"软件体系结构的可用性进行评估. "1 MD-*n* MP"可用性涉及的指标很多,如:此软件体系结构的可用性取值范围、MP 数量的最优取值、MP 冗余度对 1 MD-*n* MP 可用性的影响等.限于文章的篇幅,本文中仅以"MP 冗余度对 1 MD-*n* MP 可用性的影响"为例.

"可用性链"在 UML 图上图形化表示为一条带箭头的虚线,箭头指明了消息传送的方向.选用"AvaliabilityChain"进行建模分析时需要软件开发人员按照 AvaliabilityChain 的规范为 UML 图中的各个单元添加标签,并为标签赋值.如图 6 所示的消息序列图是"1 MD-*n* MP"的核心功能,因此,对此图进行建模分析可以保证分析结果不失一般性.添加了标签并完成了标签赋值后的"1 MD-*n* MP"消息处理的消息序列图,如图 7 所示.

由图 7 导出"1 MRD-*n* MP"的可用性链模型如图 8 所示.由图 8 可知,

AvailabilityChains[0].*availability*=*AvailabilityChains*[0].*nodes*[0].*availability*+*AvailabilityChains*[0].*nodes*[1].

Availability−*AvailabilityChains*[0].*nodes*[0].*availability*×*AvailabilityChains*[0].*nodes*[1].*availability*=

$P_{MD1\_0}+P_{MP1\_0}-P_{RD1\_0}P_{MP1\_0}$.

由于在“1 MRD-*n* MP”架构中 MD 的功能比较简单,所以可以认为此模块发生故障的概率要远小于 MP 发生故障的概率,即 $P_{\mathrm{MRD}}<<P_{\mathrm{MP}}$,所以上式可以简化为

$$AvailabilityChains[0].availability=P_{MD1\_0}+P_{MP1\_0}-P_{RD1\_0}P_{MP1\_0}\approx P_{MP1\_0}.$$

当同一模块中的所有消息处理模块(MP)具有完全相同的功能时,如果剩余的可用模块可以接替那些不可用模块的工作,则认为系统仍然可用.因此,决定系统是否可用的不是一个子系统中究竟有多少模块不可用,而是子系统中不可用的模块占总模块数目的比例关系.

套用 AvailabilityChain 样式,得出“单消息分发-多消息处理”软件体系结构的系统故障概率 P_{UA} 为

$$P_{UA}=\frac{M!}{(rM+1)!(M-rM-1)!}P^{rM+1},$$

其中 r 是系统的冗余度,即冗余模块数量与模块总数的比例关系(m/M).在得出上述针对系统故障概率的数学模型之后,就很容易完成对“单消息分发-多消息处理”软件体系结构的可用性分析.限于本文篇幅,不再对具体的分析内容加以介绍.通过基于 AvailabilityChain 建模样式对“MP 冗余度对 1 MD-*n* MP 可用性的影响”的评估可以看出:软件开发人员无须了解复杂的系统建模理论和数学理论,而仅需针对特定建模样式完成标签的添加和赋值,就可以由 UML 图生成相应的评估模型.建模样式的引入极大地降低了生成软件非功能属性评估模型的复杂度和工作量.

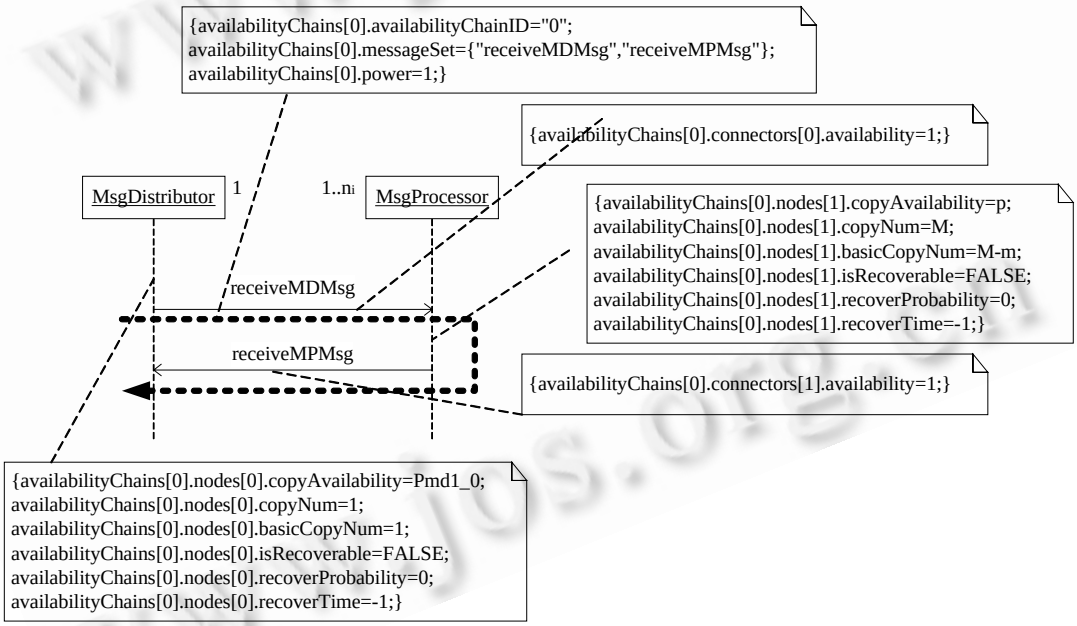

Fig.7 Message sequence chart of “1 MD-*n* MP” with AvailabilityChain applied on

图 7 基于 AvailabilityChain 的“1 MD-*n* MP”(继承关系)消息序列图

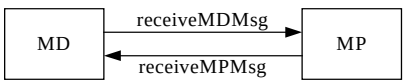

Fig.8 Availability model of “1 MD-*n* MP”

图 8 “1 MD-*n* MP”的可用性模型

从上述对 AvailabilityChain 建模样式应用的介绍中可以发现:在建模样式应用中,特定建模样式的应用场景往往会受到限制.实际上,在建模样式的应用中,往往会针对各种场景提供不同的建模样式.但这就直接导致了一系列问题:(1) 如何从众多建模样式中选取合适的建模样式;(2) 在确定了要使用的建模样式之后,如何在

UML 模型中应用此建模样式;(3) 在UML 模型中应用了建模样式之后,又应如何确定建模样式中的各个标签的取值.本文引入了扩展的样式系统——建模知识库以解决这些问题.

3 扩展的样式系统:建模知识库

样式并不是孤立存在的,它们之间互相依赖.因此往往以“样式系统”的方式将各种样式组织在一起.“样式系统”是一个样式的汇集,它包括样式在整个软件过程中实际使用的指南、描述样式如何互相联系和互相补充、支持在软件开发中高效的使用样式[8].另一方面,从 UML 的角度而言,“建模样式”表现为附加在 UML 图上的一系列标签.而标签则是附加在 UML 要素上的一系列参量.因此,在建模样式的具体应用中,非常重要的一点就是如何确定这些参量在具体的软件体系结构非功能属性分析中的取值.同时,由于对软件体系结构的非功能属性分析应在软件设计前期完成,在此阶段中,软件的实现代码还远未出现,不可能通过测量的方式获得建模必要的数据,因此,必须有相应的机制使软件体系结构设计人员可以在软件设计前期获得参量的取值,至少可以获得估计某些参量取值的依据.“建模知识库”的引入正是针对上述需求,一方面提供样式系统的能力,软件设计人员可以使用此知识库以选取最合适的建模样式;另一方面,此知识库还要管理和维护建模样式需要的各种数据信息,提供必要的“路标”,协助软件分析人员在多个选项中选取最适宜的标签取值.

3.1 建模知识库的构建与更新

建模知识库的构建和更新包括两个过程:一个是样式系统形成和演进的过程;另一个是建模中所需数据逐步积累和更新的过程.样式系统形成和演进的过程与一般的样式系统形成与演进的过程基本一致,主要涉及样式系统分类标准的确定、样式分类图式的确定、样式选取规则的确定、样式采集、样式向样式系统的集成、样式系统更新等一系列内容.限于本文篇幅,这里不再赘述.下面将重点介绍建模知识库对样式系统的增强,即建模中所需数据积累和更新的过程.

建模所需数据主要来源于以下几个方面:

- 行业经验数据,这部分内容可能源于特定行业的工程实践、行业的标准/规范等.
- 以前项目的实测结果.这部分内容是建模数据的重要来源,因为一个软件组织的历史数据最能反映此组织承担的新软件项目可能出现的情况.当然,软件组织先后承担的多个软件项目的差异,一般会导致旧有项目中获得的经验数据难以直接应用于新的软件项目中,这就需要通过类比等方式从现有数据中推出本项目中使用的数据.需要指出的是,类比方式本身不可能保证推出的数据有较高的准确度,所以,基于类比数据进行的系统分析给出的结果一般是一个“似然空间”,即数据以一定的概率落在此空间.
- 建模知识库中,建模参考数据的更新过程与软件过程紧密结合,在软件过程的各个阶段,软件技术人员都可以与建模知识库交互,从中获取所需的数据信息或是更新原有的数据信息.

3.2 建模知识库的使用

建模知识库是支持建模样式应用的最为重要的技术.软件开发人员在完成了对软件体系结构 UML 模型的设计之后,可以按照如下步骤使用建模样式:

1) 将建模知识库作为样式系统,利用其提供的分类图式检索适合的建模样式.一般来说,可以使用“UML 图类型(如交互图)”和“待分析非功能属性(如可用性)”作为样式图式的分类标准检索建模样式;
2) 依据建模知识库中维护的建模样式,这里主要是指 MdlSolution.model()中规范的一系列标签,为 UML 模型中的各个要素“粘贴”标签;
3) 使用建模知识库中维护的参量值为各个标签赋值;
4) 依据建模知识库中维护的建模样式的数据——这里主要是指 MdlSolution.tir()中规范的一系列规则,检查各个标签取值的完备性和有效性;
5) 使用建模样式生成针对特定非功能属性的分析/评估模型;
6) 使用非功能属性分析/评估模型确定软件体系结构的非功能属性指标.

4 总结及下一步工作

软件体系结构非功能属性建模技术与模型分析技术都不是本文的创新.本文的创新在于:提出了“建模样式”,通过样式技术将建模技术和模型分析技术结合在一起,同时,由于本文提出的建模样式基于 UML 扩展实现,并以“建模知识库”进一步提高了建模样式技术的实用性,从而使本文提出的软件体系结构非功能属性评估技术可以为软件开发人员所用,融入高质量软件开发过程中.

本文中提出的建模知识库是支撑建模样式在软件工程实践中应用的重要技术,其作为一种扩展的样式系统,不但要完成样式系统的功能,而且要维护和管理建模样式应用中需要使用的标签参考值.建模知识库不是一成不变的闭合系统,而是一个不断更新的动态系统.同时,建模知识库的形成与更新并不是一个独立过程,这个过程将与软件过程紧密结合.一般来说,软件开发过程模型是软件开发的宏观过程模型,反映了软件项目实施的总体过程思路[9].建立与建模样式、建模知识库配合应用的软件过程模型,将进一步促进基于建模样式的软件体系结构分析和评估的应用.

同时,本文只是给出了建模样式的 UML 描述,并没有明确给出建模样式与建模知识库交互信息的形式化描述以及建模知识库中维护信息的形式化描述.这些形式化描述将是支持软件体系结构非功能属性自动评估的技术基础之一,因此,完善建模样式应用中的形式化描述将是下一步研究中的一个重点问题.

References:

[1] Sun CA, Jin MZ, Liu C. Overviews on software architecture research. Journal of Software, 2002,13(7):1228−1237 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/13/1228.pdf

[2] Bass L, Clements P, Kazman R. Software Architecture in Practice. Addison-Wesley, 1998.

[3] Bass L, Clements P, Kazman R, Northrop L, Zaremski A. Recommended best industrial practice for software architecture evaluation. Technical Report, CMU/SEI-96-TR-025, 1997.

[4] Liliana D, Eila N. A survey on software architecture analysis methods. IEEE Trans. on Software Engineering, 2002,28(7):638−653.

[5] Pooley R, King P. The unified modelling language and performance engineering, software. In: Govan S. Proc. of the IEE Software. Stevenage: IET, 1999. 2–10.

[6] Gamma E, Helm R, Johnson R, Vlissides J. Design Patterns Elements of Reusable Object-Oriented Software. Boston: Addison-Wesley, 1994.

[7] Booch G, Rumbaugh J, Jacobson I. The Unified Modeling Language User Guide. Boston: Addison-Wesley, 1999.

[8] Buschmann F, Meunier R, Rohnert H, Sommerlad P, Stal M. Pattern-Oriented Software Architecture: A System of Patterns. Chichester: John Wiley & Sons Ltd., 1996.

[9] Renato L, Della V, Farley S. The role of software process improvement into total quality management: An industrial experience. In: Proc. of the 2000 IEEE. 2000. 29−34.

附中文参考文献:

[1] 孙昌爱,金茂忠,刘超.软件体系结构研究综述.软件学报,2002,13(7):1228−1237. http://www.jos.org.cn/1000-9825/13/1228.pdf

徐鹏(1964－),男,内蒙古呼伦贝尔人,博士,讲师,主要研究领域为下一代网络技术,通信软件.

杨放春(1957－),男,博士,教授,CCF 高级会员,主要研究领域为下一代网络技术,通信软件.