基于RUP的系统动态行为描述方案*)

RUP-Based Description Scheme of System Dynamic Actions

李 航 郭敬林 刘西洋

(西安电子科技大学软件工程研究所 西安710071)

Abstract How to describe system dynamic behaviors clearly with UML is an urgent problem to solve. This paper presents a RUP-based description scheme and validates it in the Fault-tolerant Dual Computer System.

Keywords UML, RUP, Description scheme, Dynamic behaviors, Dual computer Fault-tolerant

1 UML简介

UML最初是由Grady Booch, Jim Rumbaugh, Ivar Jacobson共同提出[1],它结合了OOA/OOD,OMT,OOSE等方法,并且为了增强其对模型动态行为的描述能力,引入了诸如Statechart, Sequence diagram, Collaboration diagram,并于1997年被OMG(Object Management Group)正式确定为国际标准,即UML 1.0。此后UML进入了快速发展的阶段,并相继在1998和1999年提出了UML 1.2和UML 1.3标准。

与此同时UML在各个领域中得到了广泛的应用,仅在国内,中国石油天然气公司开发局、地球软件公司、石油行业、中科院软件所、大唐电信、深圳中兴、国防科工委系统等均采用了UML作为系统建模语言。其中比较著名的成功案例有:四川托普集团西部软件园与中科院软件所的地税业务电子化系列软件、大唐开发的"九七工程"计费系统、深圳中兴通信公司"邮电通信设备"项目等。

2 问题的提出

以往信息系统领域中的建模以描述系统静态行为的E-R图为主,所以一开始UML的语义就很松散,非常适于建模企业信息系统。虽然UML使用了Collaboration Diagram、Statechart Diagram和Sequence Diagram[2]来描述系统动态行为以增强对并发实时系统的支持,但毕竟以上三种Diagram都是借鉴别人的,并非UML所独创,而且以上三种视图与UML的Class Diagram—静态视图的结合并非完美无缺。这样就造成了一系列在表达系统动态行为上的问题。具体表现在:

(1)如何让UML所表达的领域模型使不懂计算机的领域专家与软件工程师之间达成一致。

(2)如何使领域分析阶段产生的Use Case图对以后的分析设计起到真正的指导作用,而非一张只是用来表达系统功能的简图。

(3)在注重系统内部进程与算法的交互的系统中,如何在设计阶段描述算法和进程间的消息通信机制,而不是简单的Activity图和Sequence图。

为此,我们在实践的基础上,提出了一套方案以解决上述问题。

3 描述方案

没有一种描述方案能够适用于所有的系统,所以我们将系统分为两大类。第一种:系统功能主要由若干种业务交易组成的商业系统(如银行,保险等等),每种业务交易通过系统的Actors和系统之间若干次的交互完成,不同的交互成为不同的Use Case,若干Use Case组成一种业务,系统的功能需求通过这些Use Case能够完整地描述。

第二种:Actors和系统的交互相对较少,而Actors和系统的一次交互中,内部的数据处理过程依赖于复杂的算法和进程的交互协议。

对于第一种,Rational Rose公司的RUP[3](Rational Unified Process)已给予较好的解决。而对于第二种,这种系统常常是一些动态性很强的系统,强调进程间的相互通信及算法。这恰恰是UML的弱项。

我们的方案主要针对第二类系统,该方案虽然基于RUP,但对RUP做了重要的改进,即使用了混合建模的思想。

3.1 RUP Inception阶段

在RUP的Inception阶段中,重要的标志性成果之一就是:Use Case Model中主要Use Case及Actor的认定,而在描述Use Case与Use Case之间的交互时不能简单地使用association将两个Use Case连接起来,因为一个Use Case是一个事件流,将两个事件流用Association联系起来就表示一个Use Case中的每一个事件将和另一个Use Case中的所有事件发生关系,从而造成状态空间的急剧膨胀,造成语义上的错误。所以应认定两个Use Case之间的通信对象,并将该对象认定为Actor。在得到Use Case Model之后,进行Use Case的初步分析,将该Use Case Model按子系统的方式归类分解,即将整个系统按子系统的方式分层支解,形成一棵抽象树,该树的每个节点对应一个子系统,每个节点的孩子是对该节点对应子系统的进一步细化,并将各子系统用一个Use Case表示,以便表达该层子系统的Use Case与下层子系统之间的交互。在将系统分层支解的过程中,应注意将算法独立出来,表示为一个Use Case。这样就形成了一个完整的Use Case Diagram。

当然在具体表达时不应在一张Use Case Diagram中展现所有的层次,否则容易使人疑惑。解决这个问题的方法是将该树状图映射到相应的Use Case Diagram,以分层的方式逐步描述出该Use Case Diagram,如图1所示。

*)"九五"军事预研项目的子项目并受其资助。李 航 研究生,研究方向:分布式处理系统。郭敬林 博士生,研究方向:软件工程、数据库技术。刘西洋 博士,副教授,研究方向:软件工程、分布式处理系统。

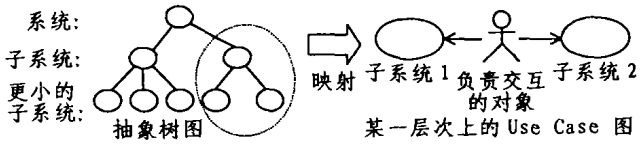

图1

现在,在 RUP 使用的 Rational Rose 工具中,就提供了这样的映射。

另外在该阶段中,Use Case 可能不能充分表达系统的整个特性。这时可考虑结合其它的各种视图将其表达清楚,而不应仅限于 Use Case 图。比如可以使用 RUP 中的 Use Case Realization 等手段将其表达清楚。

3.2 RUP Elaboration 阶段

在 Elaboration 阶段,应对领域分析得到的 Use Case Model 进行进一步的分析,从而得到 Analysis Model。虽然 RUP 将 Analysis Model 列为该阶段的可选标志性成果,但为了良好的可跟踪性,标志性成果中应有相应的 Analysis Model。在认定 Analysis Class 时,可将 Use Case 图中负责各个子系统间通信的对象认定为实体类或边界类。再通过这些实体类和边界类以及子系统的 Use Case Realization,从而认定各个具体的 Design Element,并建立相应的 Design Model。

对于领域分析阶段分离出的算法 Use Case 如何再进行细化的问题应进行具体分析。

一个算法常常涉及到交互,虽然 RUP 中对交互的描述使用了诸如:Collaboration Diagram、Sequence Diagram、StateChart、Capsule、Protocol 等手段,但这些手段对复杂的系统交互支持不够,甚至像 Capsule、Protocol 这种重要的表述手段在 Rational Rose 工具中得不到支持(仅在 Rational Rose Real-Time 中才有)。为此在实践的基础上,我们采用了混合建模的思想,即对于一种算法描述不能拘泥于 UML 所提供的那几种方式,而是应结合领域特征,采用该领域中经典的描述方式,即混合建模。比如:在工程计算领域中就使用 Metalab 进行描述。不过这就带来了一个问题:即如何将 RUP 与特定领域对算法的描述结合起来。通过实践,我们认为采用如下算法描述框架(有 Rational Rose 工具的支持),可较好地将 RUP 与特定领域对算法的描述结合起来。现描述如下:

对算法依赖性很强的系统一般都会呈现出数据流变换的特征。即在数据处理过程中,有多步的加工,每一步加工计算复杂度不同,若涉及到数据融合则加工的计算复杂度更大,从而形成整个系统的瓶颈。所以,应使用独立的进程来表示算法,使得整个系统在对数据的处理上达到平衡。另外,在决定运行时刻算法进程的个数时,需要考虑所采用算法的特性:如果所采用的算法本身是串行的,那么算法进程只能有一个,才能保证算法严格的时序性,从而保证处理的正确性;如果所采用的算法是并行的,可以考虑用多个算法进程并发地进行处理,以达到更高的效率。

使用独立的进程来表示算法,其基本的处理原则是:

提取负责对每一种数据报文的处理流程进行控制的类;

将每一种数据报文的各步处理抽象成不同的类;

对数据报文处理流程进行控制的所有类有统一的父类(下面称其为 Controller 类),有统一的接口;

所有数据报文的各步处理的提供者有统一的父类(下面称其为 Worker 类),有统一的接口;

Controller 类实现报文处理的接口,Controller 类聚合 Worker 类;

每一种报文的 Controller 和 Worker 之间通过工作区(下面称其为 Workspace 类)进行数据传递,这样做的好处是使得不同的 Worker 类接口一致。其结构框架如图2所示。

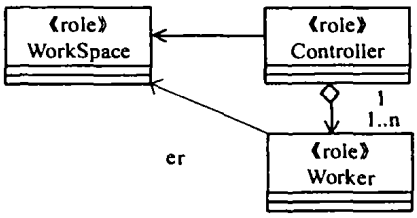

图2

上述框架因实现了数据报文处理模块的接口,并使得模块具有很高的封装性,因此,在特定领域的算法描述实现时,很容易嵌入其中;另一方面,该框架不仅使得各步处理者之间相对独立,而且使得流程控制者和各步处理的提供者之间相对独立,在一定程度上允许一个类的改变不影响其它的类,因而数据报文处理模块具有很好的封装性、可重用性和稳定性。

接着以 RUP 的 artifact 的形式给出上述算法框架与特定领域中算法的描述。

对进程之间的消息通信机制进行建模时,将消息传递机制抽象成类,当进程需要某种消息传递机制时,在类图中让代表该进程的活动类依赖代表该消息传递机制的类,在 Collaboration 图中让代表进程的活动类的实例之间通过代表消息传递机制的类的实例进行消息传递。同样以 RUP 的 artifact 的形式给出其与特定领域中对协议的描述。

对于 RUP 阶段中的 Construction 和 Transition,因为它们的主要标志性成果是对前面两阶段成果的细化及总结,这里就不再赘述。

以上方案可以较好地解决本文开始提到的三个问题。其中问题1与问题2有一定的联系,因为在以往的很多 Use Case 图中,它所表现的仅仅是一张用来表达系统功能的简图,这样虽然领域专家可能比较清楚地了解整个系统,但无法为软件工程师在以后的设计中提供指导性的帮助,所以领域专家与软件工程师之间无法达成一致。在上述方案中,通过层次化的 Use Case Diagram,能够使领域专家清晰地掌握该系统的整体结构,而且更为重要的是通过认定子系统之间交互的中介者,使得 Use Case Diagram 反映的是系统之间的交互,而不是系统功能的简单罗列,这样就更容易被软件工程师所理解;通过对负责子系统及子系统之间交互的中介者的认定,使得在 Elaboration 阶段易于提取 analysis model,易于分清系统的各个模块;通过 Elaboration 阶段对 Analysis Mode 的认定,使得系统模型拥有良好的可跟踪性;通过混合建模,使领域专家更为精确地描述该领域的行为特征,摆脱了 UML 的局限性;通过算法描述框架的认定及将进程之间的消息传递机制抽象成类,将 RUP 与特定领域对系统动态行为的描述较好地结合了起来。

4 举例

以上方案在表述 C3I 系统中的双机热备份子系统中得到了成功的应用。该 C3I 系统是国家95军事预研项目,而且截止到2001上半年,国内唯一一个使用面向对象的思想及 RUP

实现的C3I系统。由于篇幅有限，仅给出部分重要的Model，现描述如下：

i)算法框架：如图3所示。

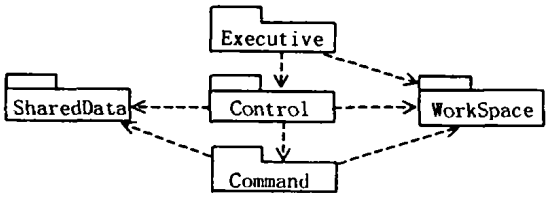

图3

ii)算法的嵌入：在control package中使用图4方式进行对算法的实现。其中CDispatcher是CListenController的worker，这些类中包含了特定领域中对协议描述的实现。

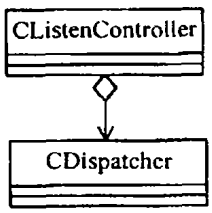

图4

结束语 UML不是万能的，它对某个专业领域的描述不可能做到像该领域中的经典描述工具那样精确，比如：工程计算中常常使用的Metalab，电信中的SDL以及为描述动力系统的计算机模型而使用的simulink。而像C3I这样大型的系统常常是多种数据模型并存，比如在服务器端常常是数据流，而服务器与席位之间是分布式MVC。所以单一依靠像Rational Rose这样的工具来表达这样大型的系统是不合适的。另外，不能过分强调可视化描述，因为在许多领域中，公式化表达算法要比图形化表达算法要强得多。所以，本文以混合方式建模为指导思想。这样做有两个好处：i)能够准确表达领域特征。ii)有助于领域专家与软件工程师达成一致。

另外本文还隐含了另一个观点：即从交互的角度进行建模，好处是容易反映系统的行为特性，在这一点上类似于OORAM[4]（Object-Oriented Role Analysis Modeling）。OORAM强调系统的交互，但它是以角色建模，从而造成角色模型的组合(role model synthesis)问题。(在OORAM中，描述一个系统主要从各个不同的侧面进行描述，从而得到多个role model，可能造成一个对象在不同的role model中担当不同的角色，这样导致的直接后果就是如何将这些role Model重新组合以描述系统整体)。OORAM也提供了角色的组合的手段，但仍然不够。

我们的方案能够较好地解决本文开头提出的问题，但由于UML本身对系统动态行为的描述限制，一些问题仍然不能得到良好的解决，具体表现在：

i)设计模式对于一个大型系统的设计是非常重要的，但在UML中只能用类的Stereotype一角色来描述，由于设计模式往往涉及协议和算法，而UML对协议和算法的建模支持异常薄弱，因而设计模式的这种UML描述实际上只能是一种文档性的描述。至于对设计模式的施用，UML就更无任何支持了。

ii)虽然可用Statechart图来描述协议，用Activity图来描述算法，但其语义是模糊的、非形式的，不足以对系统的行为进行精确的验证。

iii)本文所实现的算法框架在与特定领域中对算法的描述的结合上仍缺乏工具支持。

解决上述三个问题的方案就是对UML进行改造和扩充，提供一个类似于I-Logix Statemate的工具。这也是下一步的工作方向。

参考文献

1 Bluhme S. The History of UML http://www.iis.ee.ic.ac.uk/~frank/surp00/article1/sb398/

2 Jacobson I, Booch G, Rumbaugh J. The Unified Modeling Language User Guide. Reading Massachusetts: Addison-Wesley Longman Inc, 1999

3 Rational Unified Process 2000. Rational Software Corp.

4 Reenskaug, Wold, Lehne: Manning Working With Objects, Prentice Hall 1996

5 Statemate Magnum. http://www.ilogix.com/products/magnum/statemate_magnum.pdf

6 Introduction to Simulink. http://www.ece.wpi.edu/courses/es3011/sim/simulink.html

7 OMG Unified Modeling Language Specification, version 1.3: OMG June 1999

8 Latella D, Majzik I, Massink M. Towards a Formal Operational Semantics of UML Statechart Diagrams. In: Third Intl. Conf. on Formal Methods for Open Object-Oriented Distributed Systems, Kluwer Academic Publishers, ISBN 0-7923-8429-6, 1999. 331～347

9 Erich Gamma, Pearson. Design Pattern: Elements of Reusable Object-Oriented Software, Addison Wesley

10 Dedene G, Maes R. On the integration of the Unified Process Model in a framework for software architecture: PrimaVera Working Paper Series, Dec. 2000

11 Schleicher A, Westfechtel B, Jager D. Modeling dynamic Software processes in UML. http://citeseer.nj.nec.com/23646.html

12 Spence I, Probasco L. Traceability Strategies for Managing Requirements with Use Cases, Rational Software Corporation

13 Ambler S W. Enhancing the Unified Process. Ronin International Inc March. 2000

14 Pfleeger S L. Software Engineering: Theory and Practice, Prentice Hall

15 Pressman R S. Software Engineering: A Practitioner's Approach, Mcgraw- Hill