USDP 分析设计中基于排队网络模型的软件性能预测方法

徐忠富[1,2] 陈永光[1] 杨建华[2] 彭 珲[1]

(中国人民解放军 63880 部队 洛阳 471003)[1] (国防科技大学电子科学与工程学院 长沙 410073)[2]

摘 要 在统一软件开发过程(USDP)中,基于统一建模语言(UML)的模型是面向软件功能实现的。为了在软件开发的早期预测软件性能,基于 UML 的软件模型需要被扩展,增加获取和描述软件性能预测模型变量所需的信息。本文提出了在 USDP 分析和设计阶段预测软件性能的过程框架;定义了一个排队网络模型(QNM)元模型;基于 QNM 元模型,扩展软件分析和设计模型,增加软件系统应用模式描述以及协同实现软件系统功能的物理计算设备的特性和行为描述;采用基于可扩展标记语言元数据交换(XMI)的模型变换技术,生成软件性能预测 QNM;通过分析所生成的 QNM,可以评估和比较软件体系结构和软件设计对待实现的软件系统性能特性的影响。通过示例展示了所提出方法的可应用性。

关键词 统一建模语言,统一软件开发过程,软件性能预测,排队网络模型,元模型,可扩展标记语言,基于可扩展标记语言的元数据交换

Queuing Network Model-based Software Performance Prediction During Analysis and Design in USDP

XU Zhong-Fu[1,2] CHEN Yong-Guang[1] YANG Jian-Hua[2] PENG Hui[1]

(Unit 63880, PLA, Luoyang 471003)[1]

(College of Electronic Science and Engineering, National University of Defense Technology, Changsha 410073)[2]

Abstract UML-based software models in USDP are functionality-oriented. For the purpose of predicting software performance in the early stage of software development, they should be extended with information required for deriving and specifying variables of performance prediction models. We present a framework for predicting software performance during analysis and design in USDP. A QNM-Metamodel is defined to provide the basis for extending the analysis and design models with specifications about system usage patterns, properties and behaviors of physical computer devices that collaborate to execute the system functionality. By means of XMI-based model transformation, QNMs are generated. Analysis results of the QNMs enable the evaluation and comparison of architectural decisions and design solutions with reference to their impacts on performance properties of the software system to be implemented. A case study demonstrates the application of the proposed approach.

Keywords Unified modeling language (UML), Unified software development process (USDP), Software performance prediction, Queuing network model (QNM), Metamodel, Extensible markup language (XML), XML-based metadata exchange (XMI)

软件性能代表了软件系统在实现指定功能时的外部效果(例如响应时间和吞吐量)和内部效率(例如资源利用率)。差的性能会导致用户对软件产品形成不好的评价、带来昂贵的性能提升开销、降低软件产品的市场竞争力,甚至可能导致生命和财产损失。

随着规模的增大和复杂性的增加,许多软件系统由于在体系结构方面或设计方面存在根本性问题,不能达到预定或预期的性能要求。而传统的"fix-it-later"方法通常难以有效识别和解决这些性能问题[1]。一种可行的办法是采用主动式软件性能管理方法,即在软件开发的前期预测软件性能,消除性能瓶颈。

多年来,统一建模语言 UML(Unified Modeling Language)[2]被广泛用来构建和描述软件密集型系统的模型,建立可视化的模型文档。面向实时和嵌入式软件系统的开发,一系列商业 UML 工具(例如 IBM Rational RoseRealTime 和 ARTiSAN Real-time Studio)支持可执行软件代码的自动生成以及 UML 模型运行。其中,UML 模型运行为主动式软件性能评估和控制提供了可行手段。然而,对于更广泛领域内的软件开发,主流商业 UML 工具(例如 IBM Rational Rose /XDE, Borland Together Control Center)不支持 UML 模型的直接运行。为了进行软件性能预测,实现主动式软件性能管理,需要在软件分析和设计模型的基础上构建软件性能模型。

1 软件性能预测过程框架

统一软件开发过程 USDP(Unified Software Development Process)[3]是用例驱动、体系结构为中心、迭代和增量式的工程化过程,集成了软件开发的多个方面,包括软件系统需求描述、分析、设计、实现、测试、部署、项目管理和风险消除等。在 USDP 中,软件的一个开发周期从项目管理角度被划分为 4 个步骤:初始、细化、构造和移交。每个开发步骤包括一系列

徐忠富 博士后,副教授,主要研究方向为软件工程、电子战系统建模仿真;陈永光 工学博士,研究员,博士生导师,主要从事电子战仿真技术、电子装备试验技术、信息作战运筹分析等研究。

迭代，每个迭代至少包括 5 个阶段，分别执行需求、分析、设计、实现和测试核心工作流。基于 USDP 的软件开发通常以实现软件系统用户的功能需求为驱动，软件分析和设计模型通常面向功能实现，缺乏进行性能预测模型变量的获取和描述所需的信息。为了减小软件功能模型与性能预测模型之间的差异，软件开发人员需要获得容易理解的方法和容易使用的工具，实现(1)以受控制的方式扩展软件功能模型，增加构建性能预测模型所需的性能相关信息；(2)高效率地生成和评估性能预测模型。针对这一需要，本文提出了一个在 USDP 的分析和设计阶段预测软件性能的过程框架，如图 1 所示。

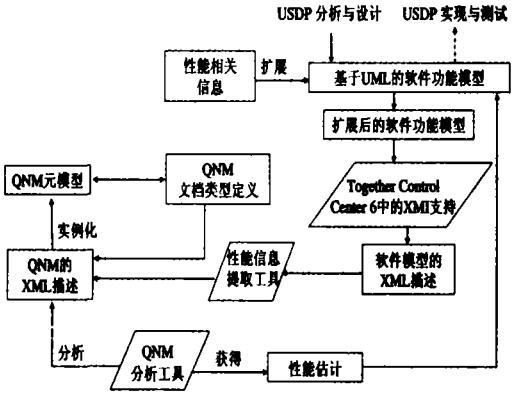

图 1　基于 QNM 的软件性能预测过程框架

图 1 所示的过程框架的核心是排队网络模型(QNM)元模型，它定义了用来描述 QNM(Queuing Network Model)[4]的语言元素。在 USDP 实现阶段前，分析与设计阶段的功能模型被扩展，增加了性能相关信息。应用 Borland Together Control Center 6 所提供的基于可扩展标记语言 XML(Extensible Markup Language)元数据交换 XMI(XML-based Metadata Exchange)支持，自动生成扩展后的软件功能模型的 XML 描述。性能信息提取工具实现了一种从被扩展软件功能模型的 XML 描述中提取性能相关信息的算法，生成并采用 XML 描述用于性能预测的 QNM。所生成的 QNM 是对 QNM 元模型的实例化。QNM 的 XML 描述的文档结构由基于 QNM 元模型的 QNM 文档类型定义 DTD(Document Type Definition)定义。

QNM 的 XML 描述所包含的信息被用来基于特定的工具构建和分析 QNM，取得待实现软件性能属性的定量估计。如果性能估计结果没有表示出性能问题，软件开发进入实现和测试阶段。否则，软件性能分析人员和开发人员要相应地修改并重新评估软件体系结构和设计方案。这样，软件体系结构和设计方案的性能缺陷在软件的分析和设计阶段被识别和排除，软件实现和测试将以经过性能确认的软件体系结构和设计方案为基础，从而避免了在软件开发的后期进行代价高昂的性能提升工作。

长期以来，在基于从 UML 模型到 QNM 变换的软件性能评估的方法和工具方面，人们开展了大量工作。例如，文[5]提出了应用 UML 图进行软件性能建模的思想，并概念化实现了从 UML 部署图到 QNM 的映射。文[6]定义了一种基于 UML 的性能建模语言(PML)，提出了一个在基于组件分布式系统的分层软件模型基础上构建扩展的 QNM 的过程框架。文[7]所介绍的工作将添加了注释的 UML 模型变换为分层 QNM。其中，一个顶层的 UML 协作图和一个 UML 部署图被用来描述系统所应用的体系结构模式的结构特性。它们被用来建立分层 QNM 的结构，即软硬件任务和它们之间的互联。UML 活动图被用来描述系统使用模式。文[8]中定义的 UML 扩展机制被用来在 UML 活动图中增加体现性能相关信息的注释。基于特定的图形变换规则，添加在 UML 活动图中的性能注释被用来获得分层 QNM 中任务的细节描述。然而，对如何保证添加了性能注释的系统使用模式描述与面向功能的软件设计和实现模型之间的一致性，文[7]没有进一步介绍。

与上述基于软件设计模型生成排队网络性能模型的工作相比，本文所介绍的工作具有以下特点：

- 在软件开发过程的前期进行性能预测，功能模型被扩展，增加构建 QNM 所需的信息；
- 性能预测基于参与软件运行的计算机资源的特性和它们的动态行为，与软件实现密切相关，增加了性能预测结果的可信度和有用性；
- 定义了一个描述 QNM 结构和行为的元模型，为基于排队网络的性能建模过程建立了通用的基础；
- UML 既被用来构建软件功能模型，又被用来描述性能信息，只需构建、演化和维护 UML 模型，能够得到标准 UML 工具的广泛支持；
- 采用 XML 描述由基于 XMI 的性能信息提取工具生成的 QNM，为选取 QNM 工具高效取得性能预测结果提供了灵活性。

下文的第 2 节详细描述 QNM 元模型；第 3 节说明如何扩展 USDP 中的基于 UML 的软件功能模型，以增加构建 QNM 所需的信息；第 4 节介绍基于 XMI 生成 QNM 的方法；第 5 节通过一个简单示例，展示所提出方法的应用。最后总结全文。

2　QNM 元模型

在结构上，一个 QNM 是相互连接的服务中心(service center)的集合体。待处理的任务(job)在服务中心之间流动。图 2 所示的 QNM 元模型定义了 QNM 的结构和行为。

QNM 的用户产生 QNM 需要处理的任务。QNM 的用户被分类，以便每类用户(user category)所产生的任务流的强度和时间特性能够被一个任务产生过程(job generation process)一致地描述。一个开放的(open)任务产生过程描述由某一类用户所产生的任务流的持续时间(duration)和任务的间隔到达时间(jobInterArrivalTime)。这些任务由一个外部的任务源生成，依次进入 QNM，在处理结束后离开 QNM。一个闭合的(closed)任务产生过程描述由数量固定的用户产生的任务流的持续时间、用户数量(userNumber)和用户思考时间(userThinkTime)。这些数量固定的用户与 QNM 交互：提交任务，检查任务处理结果，经过思考后，提交下一个任务。

在 QNM 中，一个任务的处理(job processing)包括一系列由 QNM 服务中心向该任务提供的服务(service)。任务到达服务中心后，如果服务中心忙(被占用)则等待，否则接受服务；在从服务中心得到所需服务后，任务进入下一个服务中心或离开 QNM。服务代表了服务中心在任务处理过程中所做的处理工作。一个服务用若干参数描述，包括 timeDemand(时间需求：表示服务中心提供当前服务一次所需的时间)、repetitionNumber(重复次数：表示服务中心不间断重复提供当前服务的次数)、priority(当前服务的优先级)、lock(锁定)和 free(释放)。最后两个参数是可选的，表示某个任务要独

占一个服务中心一段时间，然后允许其它任务使用该服务中心。如果某个任务(例如任务 A)要独占一个服务中心一段时间，这个服务中心被任务 A 的某个服务锁定，在其后的一段时间里只向任务 A 提供所需服务，其它任务必须等待服务，直到任务 A 通过某个服务释放该服务中心。执行锁定操作的服务具有 lock 参数，该参数包括两部分：被当前服务锁定的服务中心的名称和从当前服务开始执行到发生锁定动作时的时间间隔。执行释放操作的服务具有 free 参数，该参数也包括两部分：被当前服务释放的服务中心的名称和从当前服务开始执行到发生释放动作时的时间间隔。

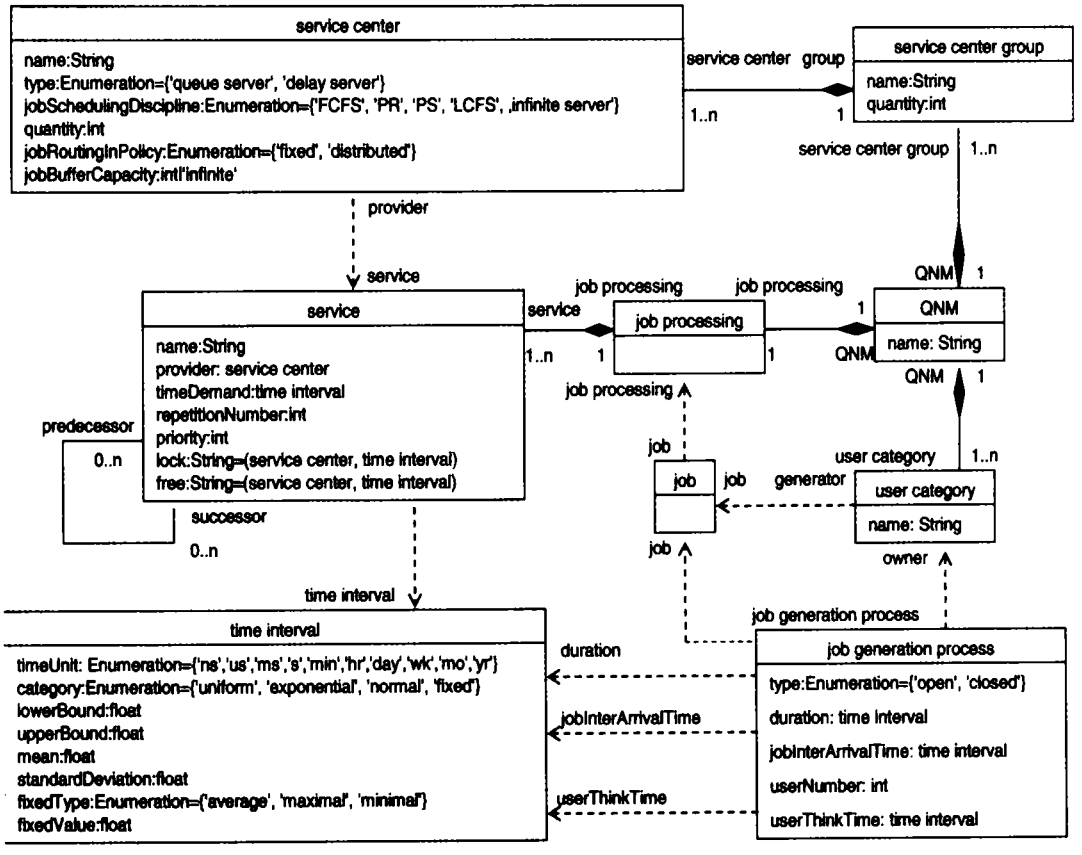

图 2　QNM 元模型

QNM 服务中心所提供的各个服务之间通过 predecessor/successor(前续/后续)关系相互关联。一个服务可以有 0 个或多个前续任务或后续任务。只有当一个服务的执行完毕后，它的后续服务的执行才能开始。

QNM 的服务中心向 QNM 中的任务提供服务，协同完成对 QNM 中任务流的处理。这里，定义两类服务中心：delay server(延迟服务器)和 queue server(排队服务器)。延迟服务器(又称为无限服务器 infinite server)没有任务队列(queue)，当任务到达时立即提供所需服务。

排队服务器包括一个任务缓存空间(即队列)以及一个或多个 server。排队服务器中的 server 数量由 quantity(数量)参数表示。任务到达排队服务器后，如果没有空闲的 server，该任务在队列中等待。当排队服务器的某个 server 空闲时，该 server 根据排队服务器的 job scheduling discipline(任务调度规则)，从任务队列中选择服务对象。排队服务器的任务调度规则可以是：FCFS(先到先服务)、PR(基于服务优先级的抢占恢复)、PS(处理器共享)、LCFS(后到先服务)等。排队服务器的 job buffer capacity(任务缓存容量)参数表示了当前排队服务器队列中所能容纳的正排队等待的任务的最大数量。

在其处理生命周期中，某个任务可能会多次到达具有多个 server 的排队服务器接受服务。因此，为具有多个 server 的排队服务器定义了一个参数 jobRoutingInPolicy(任务导入方针)，用来表示一个任务多次到达同一个多 server 排队服务器时如何选择 server。固定的(fixed)任务导入方针表示：一个任务被限定为总是由在它第一次到达当前多 server 排队服务器时提供服务的 server 来服务。分布式的(distributed)任务导入方针表示：一个任务不与所到达的多 server 排队服务器的某个特定 server 相关联，而服从该多 server 排队服务器的任务调度规则。

QNM 元模型定义了 time interval(时间间隔)元素，用来描述其它元素的与时间有关的参数，例如服务的时间需求和任务产生过程的持续时间等。时间间隔可以由概率分布函数描述，例如 uniform(均匀分布)、exponential(指数分布)、normal(正态分布)等，也可以是表示 average(平均)、maximal(最大)或 minimal(最小)的固定值。

3　软件功能模型扩展

在 USDP 中，用例(use case)捕获系统的功能需求，驱动一系列工作流，从而建立预期的软件系统行为。以实现用例为直接目标，软件系统以基于模型的方式，经历分析、设计、实现、测试等开发阶段。为了在软件系统开发的前期构建 QNM，进行软件性能预测，USDP 中面向功能的软件分析和设计模型被扩展，主要增加以下信息：(1)硬件层中实现用例所需的计算机设备的特性和行为描述；(2)用户应用软件系统的模式。扩展的具体方法如下：

· UML 用例图通常被用来表示软件系统的用例、参与

者(actor)以及它们之间的关系。其中,针对参与者与用例之间的关联,增加用户请求模式的描述,包括:用户请求的类型、持续时间、强度和时间特性。

• 软件用例实现的动态方面一般通过一个或多个 UML 顺序图来表示。这里,我们强调对 UML 所定义的消息标签(message label)[2]的使用。应用消息标签表示设计对象之间消息传递所引起的操作调用的前续/后续关系,包括操作的并发、条件和递归调用。

• 构造以预测软件性能为直接目的的附加 UML 活动图。针对每个用例,如下构造活动图:(1)顺序图中设计对象所执行的操作(operation)被映射为待构造的活动图中的可分解的活动状态(activity state),根据操作执行的时间顺序,正确建立映射得到的活动状态之间的转移(transition);(2)每个操作所代表的功能被分解为若干原子任务(atomic task)。每个原子任务仅仅由硬件层的一个计算机设备(例如处理器和 I/O 设备)执行。原子任务的执行过程不能被中断。考虑到顺序图中的操作调用嵌套(即某个操作在执行过程中调用其它操作),从操作到原子任务的分解首先从位于操作调用嵌套最底层的操作开始,即最先分解不产生其它操作调用的操作,依次向上;(3)将操作分解过程中产生的原子任务映射为 UML 所定义的不可分解的动作状态(action state),并建立它们之间的转移关系。针对每个操作,用与该操作所包含的所有原子任务相对应的所有动作状态(包括转移关系)替换活动图中与该操作对应的活动状态。与原子任务对应的动作状态的并发和条件执行分别通过成对使用 UML 活动图中的同步条(synchronization bar)和判断图标(decision icon)来表示。动作状态的递归执行通过自定义的"虚拟动作状态"(dummy action state)对来表示。虚拟动作状态不与任何原子任务对应,只是用来表示一个动作状态序列的重复执行次数;(4)描述每个动作状态所对应的原子任务的运行时特性,包括执行时间需求、重复次数、优先级、对计算机设备的独占使用情况等。对每个原子任务,指明实现其功能的计算机设备的名称。通过这种方式,活动图描述了在软件用例执行期间协同处理用户需求的计算机设备的量化行为。

• UML 部署图一般被用来描述软件运行时计算节点的配置和互连网络以及设计对象在计算节点上的分布。这里,UML 部署图被扩展,增加了在上述设计对象操作到原子任务的分解过程中所识别出的计算机设备以及它们特性的规格说明。

4 基于 XMI 的 QNM 生成

应用 Borland Together Control Center 6 所提供的 XMI-1.1-for-UML-1.3 导出机制,扩展后的基于 UML 的软件功能模型被自动转换为 XML 文档,文档的结构符合 UMLX13-11.dtd[9]。该 DTD 文件基于 UML1.3 物理元模型,遵循 XMI 规格说明中所描述的 XMI DTD 设计与生成规则。

设计并实现了一个性能信息提取工具。该工具解析描述被扩展软件功能模型的 XML 文档,抽取其中的性能相关信息,针对每个软件用例分别生成一个 QNM。软件用例图中的参与者和用户请求模式分别被映射为 QNM 的一类用户和任务产生过程。软件用户请求被映射为由 QNM 处理的任务。参与软件执行的计算机设备和它们所执行的原子任务分别被映射为 QNM 服务中心和它们所提供的服务。

由性能信息提取工具生成的 QNM 采用 XML 作为描述语言。其中,QNM 元素由一系列 XML 元素描述。描述 QNM 的 XML 文档的结构遵循 QNM-DTD。该 DTD 从 QNM 元模型导出,其树状结构如图 3 所示:节点代表了 XML 元素和它们的属性,箭头从嵌套元素节点指向被嵌套元素节点,箭头旁的数值表示每个元素的多样性,其中,1..*n* 代表 1 或多个,0..*n* 代表 0 或多个。

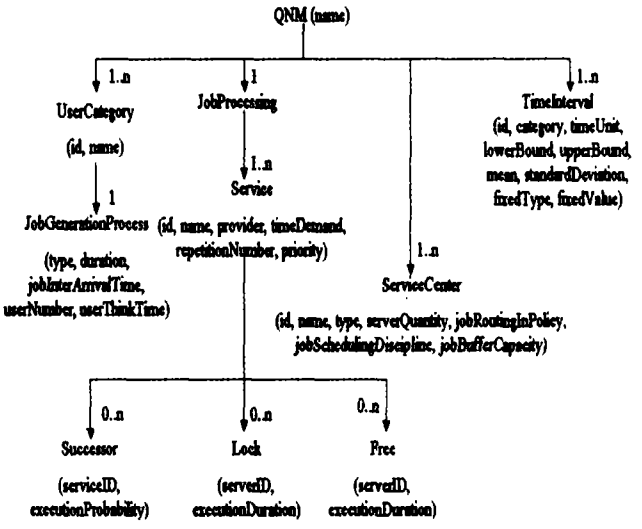

图 3 QNM-DTD 的树状结构图

5 示例

本节以自动出纳机(ATM)系统为例,展示前面提出的软件功能模型扩展和性能预测 QNM 的生成及评估方法。

5.1 软件功能模型扩展

ATM 系统被银行客户用来办理帐户信息查询、取款、存款和转帐业务。这些功能需求由四个用例表示,如图 4 所示。

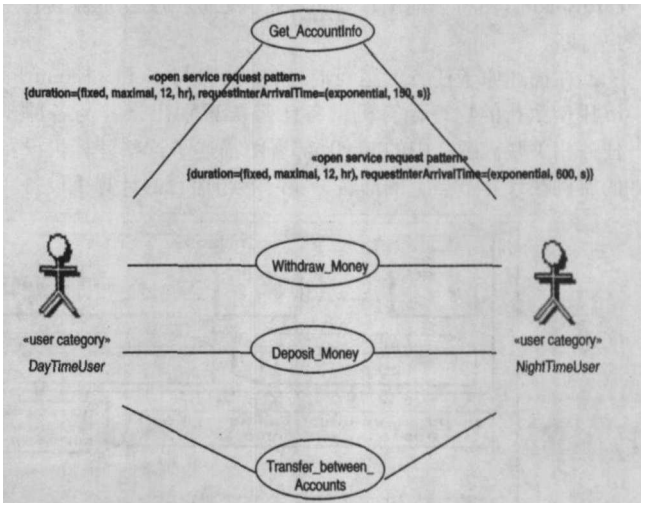

图 4 ATM 系统的用例图

本文以图 4 中 Get_AccountInfo 用例的设计和实现为例,展示面向性能分析扩展软件功能模型的方法。该用例的用户可分为两类:DayTimeUser 和 NightTimeUser,分别在白天和晚上使用 ATM 系统查询帐户信息。两类用户的请求分别由两个开放的请求模式描述。用户请求的到达间隔时间服从指数分布,均值分别设为 150s 和 600s。用户请求模式描述作为扩展信息被增加到两类用户与 Get_AccountInfo 用例之间的关联上。

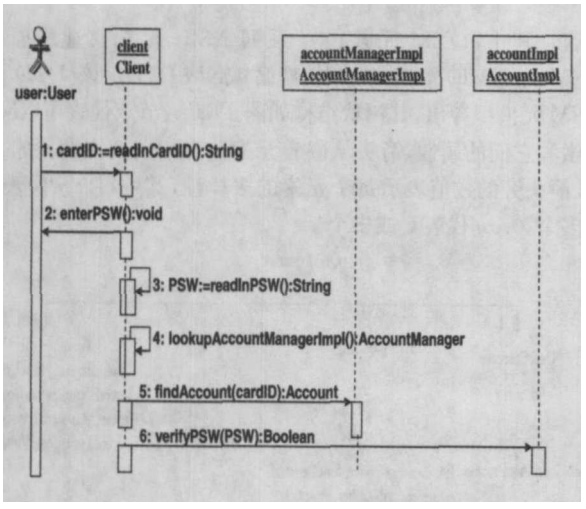

图 5　Get_AccountInfo 用例设计对象交互(片断)

ATM 系统实现采用客户服务器体系结构,采用 Java 作为编程语言。Get_AccountInfo 用例设计为由 4 个对象(即 user, client, accountManagerImpl, accountImpl)实现。图 5 给出了描述这 4 个设计对象交互的 UML 顺序图的片断(只给出了前 6 个操作调用)。

采用上述设计对象操作映射和分解方法(见前面第 3 节),构建了 UML 活动图,将 Get_AccountInfo 用例的实现表示为由物理计算设备所执行的相互关联的原子任务的集合。图 6 给出了该活动图的片断:图 5 中设计对象 client 的操作 readInCardID 被分解为原子任务 readCardID 和 getCardID;图 5 中设计对象 user 的操作 enterPSW 由原子任务 enterPassword 表示;图 5 中设计对象 client 的操作 readInPSW 由原子任务 readInPassword 表示。

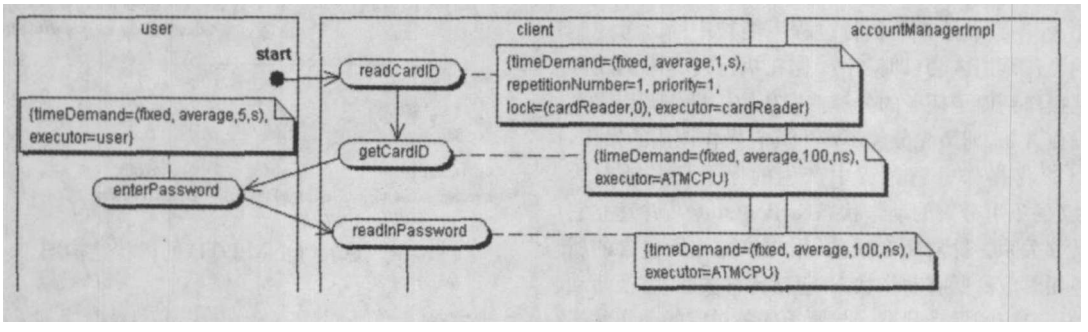

图 6　Get_AccountInfo 用例活动图(片断)

图 6 中的原子任务的运行时特性通过文本注释加以描述。例如,原子任务 readCardID 的运行时特性描述包括:该原子任务由计算机设备 cardReader 执行;平均执行时间为 1s;执行次数为 1;优先级为 1;执行开始后立即锁定 cardReader,使得当前用户与 ATM 机之间的交互不被其它用户打断。

在描述原子任务的运行时特性时,参与 Get_AccountInfo 用例执行的物理计算机设备已经被识别出来并与各原子任务相关联。图 7 中的 UML 部署图描述了这些计算机设备的实例在计算节点上的部署。实际参与运行的计算机设备实例的数量由其名称下方的数值表示。其中,ATM 的数量为 2,表示当前系统中共有两台 ATM 机。出于直观展示的目的,这里采用文本注释描述计算机设备的特性,包括 taskSchedulingDiscipline(任务调度策略)、taskBufferCapacity(任务缓存容量)等。在实际的 Together Control Center 项目中,计算机设备的特性在计算机设备的 Requirements 规格说明的 description 字段中描述。为了表示 ATM 机和 bank server 中的 CPU,定义了一个构造型 processor。该构造型有两个附加参数:MIPS 和高级语言与机器语言指令之间的比率。

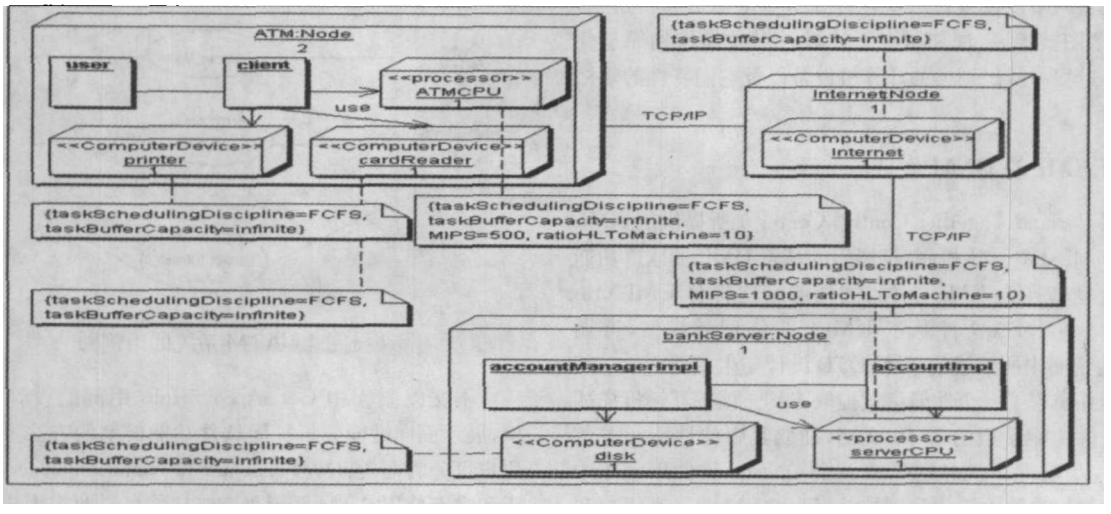

图 7　扩展后的 Get_AccountInfo 用例实现部署图

5.2　QNM 生成

上述扩展后的软件功能模型被转换为 XML 文档。性能信息提取工具提取性能相关信息,生成并用 XML 描述 ATM 系统的 QNMs。针对 Get_AccountInfo 用例所生成的开放

QNM 的拓扑结构如图 8 和图 9 所示。图 7 中六个计算机设备实例被映射为六个 QNM 排队服务器。来自两个外部任务源(即图 8 中的 source1 和 source2)的任务从 ATM1 和 ATM2(对应两台 ATM 机)进入 QNM,接受 QNM 各排队服务器提供的服务。这些服务与 Get_AccountInfo 用例执行过程中计算机设备执行的原子任务相对应。在得到所有所需服务后,任务离开 QNM,进入一个 QNM 外部的收容器(sink)。

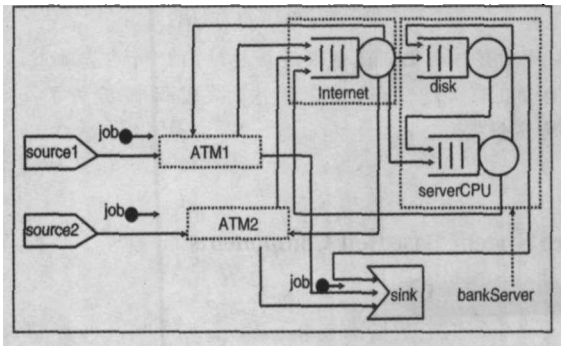

图 8 Get_AccountInfo 用例的 QNM 拓扑结构

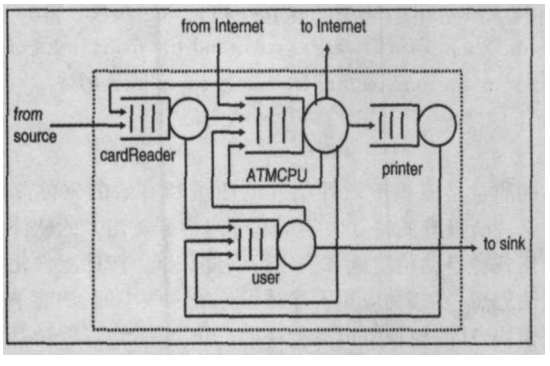

图 9 ATM1 和 ATM2 的内部结构

5.3 QNM 评估

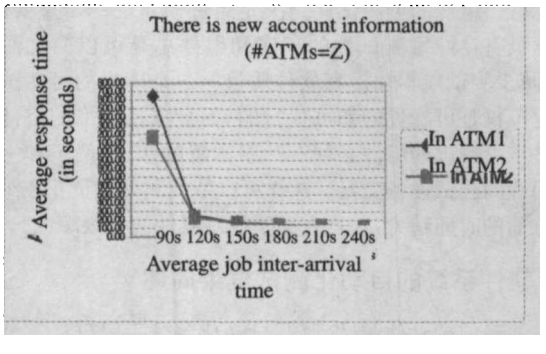

图 10 ATM 系统平均响应时间随用户请求平均间隔到达时间的变化

采用仿真工具 Simul8[10] 被用来分析和评估上述 Get_AccountInfo 用例的 QNM。每次仿真试验中当前 QNM 的仿真模型运行 15 次,每次运行的仿真时间为 43200s(即 12h)。通过分析仿真运行结果,可以评估不同体系结构和设计方案下的软件性能指标,包括响应时间、计算机设备的利用率、用户队列长度和排队时间等。例如,图 10 所示的分析结果表明,基于当前的软件设计和硬件配置,当用户请求的平均间隔到达时间从 120s 减少到 90s 时,ATM 系统的响应性显著下降。这里,响应性指的是系统的平均响应时间,针对当前查询能够获得新增帐户信息的 ATM 用户。

结束语 为了更高效地开发高质量的软件,可以在统一软件开发过程中软件开发的前期进行软件性能预测。基于 QNM 元模型,软件功能模型(主要包括用例、设计和部署模型)可以被扩展,增加软件性能相关信息(包括系统使用模式以及对参与系统功能实现的物理计算机设备的特性和行为描述)。应用基于 XMI 的模型变换,性能相关信息可被用来生成 QNM。通过对 QNM 进行分析,可以评估和比较不同的体系结构和设计方案对待实现软件性能特性的影响,指导软件体系结构和设计方案的选择和优化。

在 2.0 版本中,UML 的软件行为建模和运行环境建模等能力得到显著增强。主流 UML 建模工具对 UML2.0 的支持将变得更加完备和成熟,从而为面向性能分析扩展软件功能模型和构建基于排队网络的软件性能预测模型提供更加便利的条件。

参 考 文 献

1 Smith C U. Performance Engineering of Software Systems[M]. Reading: Massachusetts, Addison-Wesley, 1990

2 Rumbaugh J, Jacobson I, Booch G. 统一建模语言参考手册[M]. 北京:机械工业出版社, 2001

3 周伯生,冯学民,樊东平. 统一软件开发过程[M]. 北京:机械工业出版社, 2004

4 Lazowska E D, Zahorjan J L, Graham G S, Sevcik K C. Quantitative System Performance: Computer System Analysis Using Queuing Network Models[M]. Prentice Hall, 1984

5 Pooley R, King P. The Unified Modeling Language and Performance Engineering[C]. In: IEE Proceedings-Software, 1999, 146(1): 2~10

6 Kähkipuro P. UML-based Performance Modeling Framework for Component-based Distributed Systems[C]. In: Dumke R, et al. eds. Performance Engineering - State of the Art and Current Trends. LNCS 2047. Berlin: Springer-Verlag, 2001. 167~184

7 Petriu D C, Shen H. Applying the UML Performance Profile: Graph Grammar-based Derivation of LQN Models from UML Specifications[C]. In: Field T, et al. eds. Proceedings Performance TOOLS 2002: 12th International Conference on Modeling Tools and Techniques for Computer and Communication System Performance Evaluation. London, UK, 2002

8 Object Management Group (OMG). UML Profile for Schedulability, Performance, and Time Specification. OMG Adopted Specification: ptc/02-03-02. http://www.omg.org/cgi-bin/doc?ptc/2002-03-02

9 UML 1.3 DTD for XMI 1.1. UMLX13-11.dtd. http://cvs.berlios.de/cgi-bin/viewcvs.cgi/metadata server/metadataserver/resources

10 SIMUL8 Corporation. http://www.simul8.com/