从UML到GSPN的转换和性能分析方法

胡 翔[1,2] 焦 莉[1] 柴叶生[3]

(中国科学院软件研究所计算机科学国家重点实验室 北京 100190)[1]

(中国科学院大学计算机与控制学院 北京 100049)[2] (苏州大学计算机科学与技术学院 苏州 215000)[3]

摘 要 UML模型一般不能直接进行性能分析,需要利用模型转换的方法将其转换成其他分析模型,比如排队论、随机进程代数或者随机Petri网等模型。利用Eclipse平台上的Papyrus建立3种类型的UML模型(用例图、部署图和活动图)来对系统进行建模,并利用MARTE规范添加一些性能相关的信息;然后利用ATL实现UML模型到广义随机Petri网(GSPN)模型的转换,并使用XStream将上一步得到的GSPN模型转换成分析工具所支持的格式;最后利用基于GSPN的性能分析方法进行系统性能分析。同时给出了一系列性能指标的计算方法,如利用率、吞吐量、平均等待请求的数目以及响应时间等,可以考察系统性能的多个方面,方便系统设计和开发人员对系统性能进行分析和优化。

关键词 模型驱动工程,UML,Petri网,模型转换,MARTE

中图法分类号 TP311 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2016.11.009

Transforming UML to GSPN for Performance Analysis

HU Xiang[1,2] JIAO Li[1] CHAI Ye-sheng[3]

(State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing 100190, China)[1]

(School of Computer and Control Engineering, University of Chinese Academy of Sciences, Beijing 100049, China)[2]

(Department of Computer Science and Technology, Soochow University, Suzhou 215000, China)[3]

Abstract A UML model cannot be analyzed for performance requirements directly, and it should be transformed into analyzable models such as queueing models, stochastic process algebra models or stochastic Petri nets models. In this paper, three kinds of UML models(use case diagrams, deployment diagrams and activity diagrams) and suitable annotations from the profile for MARTE were chosen to build performance models by the tool Papyrus on the platform Eclipse. UML models are transformed into GSPN models by ATL, and the obtained GSPN models are further transformed into the formats that analyzers can support. At last, the performance can be analyzed by using the performance analysis method based on GSPN. Some performance metrics are given to investigate the system, including utilization, throughput, the average number of waiting requests and response time, which can be referred by system designers and develo-pers to analyze and optimize the performance.

Keywords Model-driven engineering, UML, Petri nets, Model transformations, MARTE

1 引言

在系统设计和开发过程中,不仅要求系统的功能正确,而且要求系统具有高效率和低消耗的特点,即系统具有较好的性能,性能问题越来越受到人们的关注。利用性能分析的方法可以得到一些性能相关的指标,比如利用率和吞吐量等,这些性能指标能够反映出系统内可能存在的性能瓶颈问题,通过解决性能瓶颈问题可以提高系统的性能。传统的性能分析方法主要是基于测量的,往往是在系统部署之后才开展,通过系统运行时得到的数据来反映系统的性能。而在真实系统中,性能问题可能在系统开发过程的早期已经被引入,在系统部署之后其才被发现就为时已晚,因为这些性能问题可能是由设计和架构的因素导致的,问题的解决需要设计人员和开发人员回溯到早期阶段对系统进行重新设计,这将会大大增加开发成本。因此,尽可能早地对系统进行性能分析十分必要。本文采用基于模型的性能分析方法在系统开发过程的早期就对系统建立性能模型,并分析其性能。这样,性能问题能够在系统开发周期的开始阶段被发现并解决,从而提高了系统开发效率并减少了开发成本。

统一建模语言(Unified Modeling Language, UML)[1]作为一种标准化的通用建模语言,提供了丰富的图形化建模技术,方便对复杂分布式系统(如嵌入式系统和商业流程)进行建模,被广泛应用在系统开发的过程中。MARTE规范[1]对UML进行了扩展,增加了刻画时间、资源等方面的信息,能更方便地对模型进行性能分析。MARTE规范已经取代了SPT规范[1]。

到稿日期:2015-09-11 返修日期:2016-02-23

胡 翔(1989-),男,博士生,主要研究方向为形式化方法、Petri网理论与应用,E-mail:hux@ios.ac.cn;焦 莉(1964-),女,博士,研究员,博士生导师,CCF高级会员,主要研究方向为形式化方法、Petri网理论与应用;柴叶生(1988-),男,硕士,主要研究方向为模型驱动软件工程、系统可靠性分析和压力测试。

基于模型的性能分析方法主要涉及到的模型包括马尔可夫链、排队论、随机进程代数以及随机 Petri 网等模型。随机 Petri 网具有严格的语义描述和丰富的分析方法，适合于对并发和分布式系统进行建模。随机 Petri 网在基本 Petri 网上增加了时间约束，可以进行性能分析。但是，对于不熟悉形式化方法的设计人员来说，他们很难迅速掌握这些形式化模型，而且直接对系统进行形式化建模有很大难度。另外，利用 UML 进行系统建模虽然更为方便快捷，但不能直接进行性能分析。本文提出了一种行之有效的解决办法，将 UML 模型转换成与之对应的形式化模型，这样，设计人员只需建立 UML 模型就可以对自动转换而成的形式化模型进行性能分析。近年来，这种模型转换的方法在软件性能工程中得到了广泛的应用[2,3]。

目前存在一些将 UML 模型转换成 Petri 网模型进行性能分析的研究工作，这些 Petri 网模型主要是随机 Petri 网(SPN)及其扩展——广义随机 Petri 网(GSPN)。例如，文献[4]使用了用例图、类图、顺序图和状态图对系统进行建模，然后将其转换成 GSPN 模型来进行性能分析，并通过一个通信协议的例子对这种方法加以说明。文献[5,6]提出了一种组合的方法，首先，将 UML 模型中的每个功能模块转换成一个子 GSPN 模型，然后将这些子 GSPN 模型组合成最终的模型。

活动图常常被用来刻画系统的动态行为。由于活动图与 Petri 网具有类似的图形结构，文献[7-9]利用 Petri 网对活动图的语法与语义描述进行了形式化定义，使得两者的对应关系更容易被找到。文献[10]研究了 UML 模型和 Petri 网这两种建模语言的区别和联系。文献[11,12]将活动图应用到软件性能工程的方法中。文献[13]结合了用例图、部署图、活动图来对系统的软硬件进行建模分析，利用性能上下文模型(Performance Context Model，PCM)作为模型转换过程中的一个中间模型，并通过一个音乐网站的例子验证了 UML-PCM-PN 的模型转换分析方法。

本文利用 Eclipse 平台上的 UML 建模工具 Papyrus[14]，通过建立用例图、部署图和活动图的 UML 模型来对系统进行建模，并利用 MARTE 规范添加了必要的信息，然后利用 ATL[15] 实现 UML 模型到 GSPN 模型的转换，并使用 XStream[16] 将上一步得到的 GSPN 模型转换成分析工具所支持的格式，最后利用基于 GSPN 的性能分析方法进行系统性能的分析与优化。本文给出了一系列性能指标的计算方法，如利用率、吞吐量、平均等待请求的数目以及响应时间，可以考察系统性能的多个方面，方便系统设计和开发人员对系统性能进行优化。

与文献[4-6,11,12]不同，本文将不同的 UML 模型综合在一起考虑，刻画了系统的功能划分、资源部署和动态行为，并且可以设置不同的工作量以及用户控制策略，不仅仅是从某种单一的视图出发考察系统的某一方面。不同于文献[13]，本文采用最新的 MARTE 规范，取代了 SPT 规范，从而能够为 UML 性能建模提供更好的支持，同时本文将模型转换过程分为语义转换和语法转换两部分，采用较为成熟且具有统一平台支撑的转换技术，实现了从 UML 模型到 GSPN 的自动转换，并给出了更多类型的性能指标以及计算公式，可以用来考察系统性能的多个方面。

本文第 2 节介绍如何利用 UML 模型和 MARTE 规范进行性能建模；第 3 节对 UML 模型到 GSPN 模型的自动转换过程进行了详细的说明；第 4 节给出了性能分析的方法以及性能指标的计算公式；最后对全文进行了总结。

2 利用 UML 和 MARTE 进行性能建模

本文选用 UML 模型中的用例图、部署图和活动图分别对系统的功能划分、资源部署、动态行为等关键模块进行建模。在性能建模的过程中，需要利用 MARTE 规范在 UML 模型上添加必要的信息，比如工作量约束、时间约束以及一些输入参数(容量限制、并发数限制)等。本文使用的 UML 模型以及 MARTE 规范如图 1 所示。

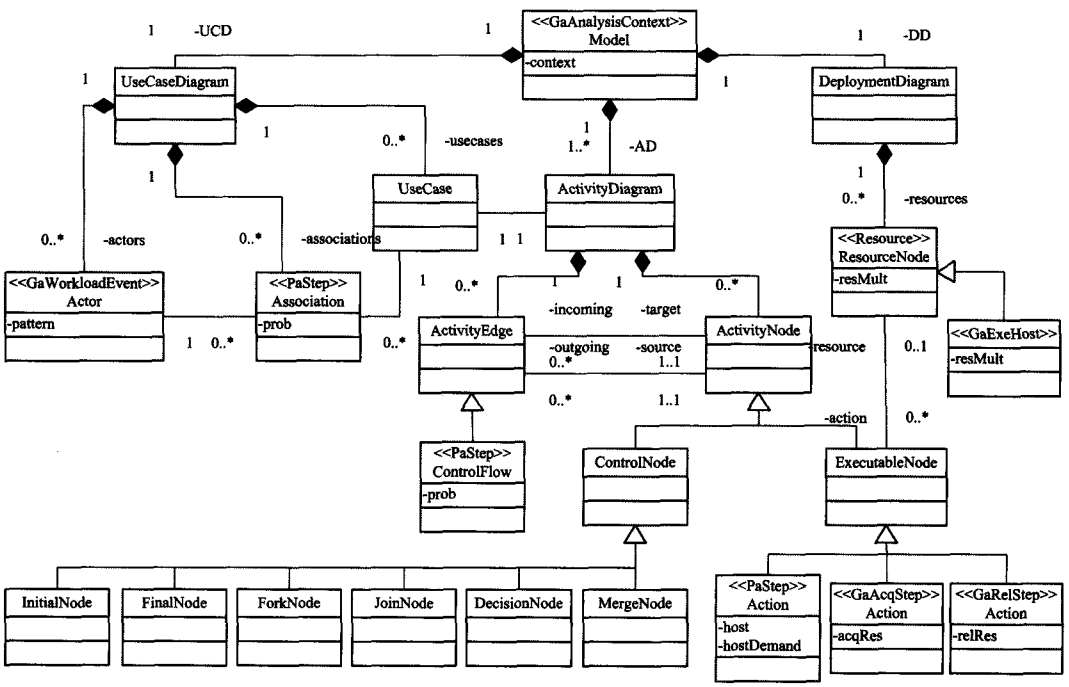

图 1 UML 模型以及 MARTE 规范

模型包含一个用例图、一个部署图和多个活动图。在模型的最顶层添加构造型 GaAnalysisContext，它包含当前场景下的上下文信息，即系统的功能划分、资源部署和动态行为，它还可以定义一些全局参数(属性 context)。

用例图包含 3 种元素：角色、关联弧和用例。同一个角色可以与多个用例关联，同一个用例也可以与多个角色关联。当同一个角色关联到多个用例时，往往是按照一定的概率，通过在关联弧上添加构造型 PaStep(属性 prob)来实现。在研究一个系统在不同场景下的性能时，需要设定某种类型的角色的数目以及到达方式(如泊松到达)，即工作量约束，这些可以通过在角色上添加构造型 GaWorkloadEvent 来实现(属性 pattern)。工作量一般包括开环和闭环两种，闭环的工作量需要给定某种类型的角色的数目以及到达方式，该角色在系统内可以循环往复地执行某些操作，不会中途离开，即系统是封闭的。相反地，开环的工作量允许角色以某种到达方式进入系统，但对角色的数目没有限制，而且该角色在系统内执行完某些操作后会离开。用例图中每个用例所代表的抽象功能是通过活动图来详细刻画的，故用例图中的用例与活动图是一一对应的。

部署图包含一个或多个资源节点，通过添加构造型 Resource 或者 GaExeHost 来表示资源的数目(属性 resMult)等信息。资源节点可以参与系统中的一个或多个动作，既可以主动地执行某个动作(GaExeHost)，又可以在某个动作的执行过程中被获取和释放(Resource)。

活动图包含活动边和活动节点，两者是关联在一起的，通过活动边和活动节点之间不同的关联关系可以刻画不同的控制结构，比如顺序、分支、并发等结构。当执行 decision 节点时，可以按照一定的概率选择某个分支执行，通过在相连的活动边上添加构造型 PaStep(属性 prob)可以设置概率。活动节点包括控制节点和可执行节点。控制节点分为 6 种，分别是初始节点、终止节点、fork 节点、join 节点、decision 节点、merge 节点。可执行节点可以与部署图中的资源节点进行关联，通过添加构造型 PaStep、GaAcqStep 或 GaRelStep 可以设置这种关联关系，分别表示该动作由某个资源主动执行、该动作需要获取和释放某个资源，同时，这些构造型还可以添加动作执行速率(属性 hostDemand)等信息。

本文采用的 UML 建模工具是 Papyrus[14]，它是 Eclipse 平台上的一个组件，被认定为 Eclipse 平台上的官方 UML 建模工具，提供方便用户建模的图形化界面，并且支持 MARTE 规范，广泛应用在模型驱动开发领域。图 2 给出了一个利用 Papyrus 建立的 UML 模型，这是一个简单的网站应用，包括一个用例图、一个部署图和一个活动图。构造型 GaAnalysisContext 定义了两个全局参数，即容量限制和并发数限制。用例图有两类角色和一个用例，角色分为注册用户和匿名用户，分别添加了闭环和开环的工作量约束，每个角色按概率 1 与用例进行交互。部署图中只有一个资源节点 Server，注释中给出了该资源的实例个数为 2。活动图表示该网站应用的执行过程，首先 Server 接受用户的请求(Acquire Request)，然后 Server 会 fork 出两个线程，同时执行两种操作：1)发送网页广告(Send Banner)；2)处理用户的请求，发送网页信息。当用户的请求合法时，Server 会发送正常的网页信息(Send Content)；否则，发送错误信息(Send Bad Request Page)。上述 4 个动作的执行者和执行速率以及从 decision 节点发出的两条弧的概率都在注释中给出。这个例子简单直观地说明了利用 UML 和 MARTE 进行性能建模的方法，利用 MARTE 规范为 UML 模型添加必要信息十分方便。

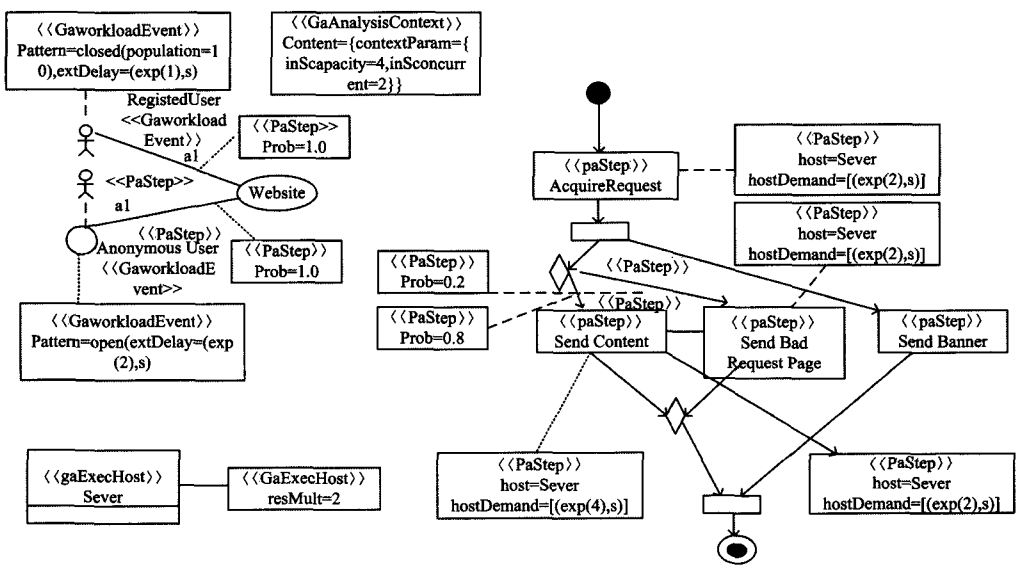

图 2　一个 UML 模型的例子

3　模型转换

模型转换广泛应用于模型驱动工程领域，它可以根据一定规则将源模型转换成目标模型，其大致过程是：首先抽象出源模型和目标模型的元模型，然后构造元模型间的语义映射，并且根据模型转换技术定义模型转换规则，从而在语义层面实现模型间的转换。经过语义转换得到的目标模型往往不能够直接被使用，还需要经过语法转换得到符合某种特定语法的模型，即分析工具所支持的模型格式。本文在语义层面上采用的转换技术是 ATL[15]，而在语法层面上采用的转换技术是 XStream[16]。

模型驱动工程中的 3 个核心概念是：模型(model)、元模

型(metamodel)和元元模型(metametamodel),区分开这 3 个概念有助于理解 ATL 转换过程。模型是对真实世界的抽象,而元模型是对模型的抽象,着重强调模型中的元素、属性和关联关系。类似地,元元模型又是对元模型的抽象,定义了元模型的语法和语义。元元模型既是自身的抽象,又是自身的实例。对一个系统进行建模时,模型要遵循元模型,而元模型也要遵循元元模型。

ATL 模型转换过程如图 3 所示,其包含了转换过程所有的模型、元模型、元元模型以及相互之间的关联关系。本文采用的元元模型是 Ecore,即 MMM_{Ecore},它是由 EMF(Eclipse Modeling Framework)定义的。在模型转换的过程中,源模型和目标模型分别是添加了 MARTE 规范的 UML 模型和 GSPN 模型,即 M_{UML} 和 M_{GSPN},根据 Ecore,需要抽象出源模型和目标模型的元模型,即 MM_{UML} 和 MM_{GSPN},分别如图 1 和图 4 所示。ATL 模型转换程序也被视为一种模型,即 M_{ATL},它依赖于 UML 元模型和 GSPN 元模型;同时,ATL 模型转换程序的语法和语义规则的描述也被视为一种元模型,即 MM_{ATL}。至此,给定一个遵循元模型 MM_{UM} 的 UML 模型,就可以通过 ATL 模型转换程序(M_{ATL}),得到遵循元模型 MM_{GSPN} 的 GSPN 模型。ATL 是一种混合的声明式和命令式的编程语言,它可以声明源模型和目标模型的元素,同时可以提供一种命令式的指令来表达元素之间的映射关系。下面将逐一介绍 UML 模型与 GSPN 模型中元素、属性以及关联关系的映射关系。

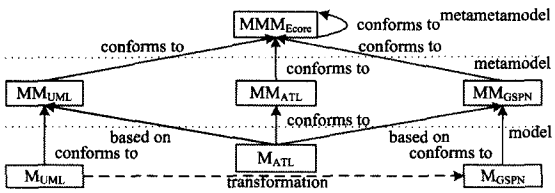

图 3 ATL 转换过程

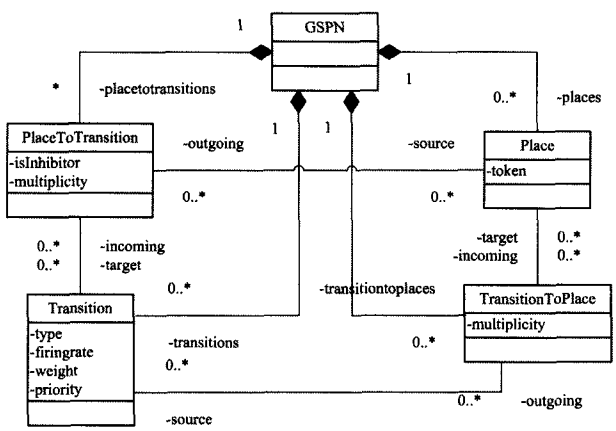

图 4 GSPN 的元模型

用例图的转换规则如图 5 所示。其中,角色根据添加的构造型 GaWorkloadEvent 分为两种类型,分别具有闭环的和开环的工作量。图 2 中注册用户具有闭环的工作量,库所 population 是由构造型的属性 pattern 中相应的参数转换而来,库所内 token 的数目等于其参数值。时间变迁 T0 表示用户到达过程,其执行速率由参数 extDelay 转换而来。到达的用户被保存在库所 P0 处,然后根据系统的访问控制策略对用户进行访问控制(T1,T2,T3)。库所 capacity 和 concurrent 表示系统的容量限制和并发数限制,是由构造型 GaAnalysisContext 的属性 context 中相应的参数转换而来,库所内 token 的数目等于其参数值。当系统容量未达到最大值时,允许用户进入(T1),否则,拒绝用户进入(T2)。进入系统的用户被保存在库所 P1 处,当系统并发数限制未到达最大值时,被允许使用系统功能(T3);否则,需要一直等到在库所 P1 处直到约束条件得到满足。T3 的发生表示用户使用系统功能的开始,同时使得库所 P2,P3 被标记。P2 相当于系统的入口,与由关联弧转换而来的 GSPN 结构相连,最终关联到活动图转换而成的 GSPN 结构。P3 表示该用户的状态,当被标记时表示该用户处在系统的访问状态。当用户使用完系统功能时,需要返回初始状态(T4),T4 与关联弧转换而来的 GSPN 结构相连,T4 的发生使得之前被消耗的 token 返回到相应的库所(population,capacity,concurrent)。图 2 中匿名用户具有开环的工作量,其转换而成的 GSPN 结构与闭环的工作量类似,只需将库所 population 以及与之相连的弧去掉即可。用例图中的关联弧被转换成两个起连接作用的 GSPN 结构,每个结构与不同的库所或变迁相连,从而将角色和活动图转换而来的 GSPN 结构连接在一起。这样,用户就可以使用关联弧指向的用例所代表的功能,并且可以在使用完某个功能之后返回到原来的位置。当关联弧上添加了构造型 PaStep 并设置其属性 prob 时,该属性值就被转换成瞬时变迁 T5 的权值,从而使得一个用户可以按照概率使用不同的系统功能。

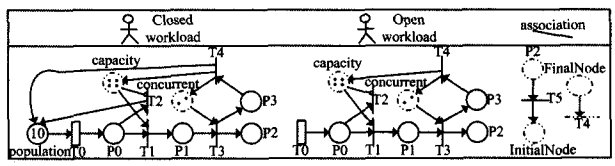

图 5 用例图的转换规则

资源图中的一个资源节点被转换成一个库所,添加的构造型 Resource 或者 GaExeHost 的属性 *resMult* 值被转换成库所内的 token。

活动图的转换规则如图 6 所示。活动图中的开始节点、结束节点、decision 节点、merge 节点都被转换成一个库所。活动图中的控制流转换成一个起连接作用的 GSPN 结构,如果弧上添加了构造型 PaStep,需要将其属性 *prob* 值转换成瞬时变迁的权值。Fork 节点和 Join 节点转换而成的 GSPN 结构是对应的,可以支持两个或者两个以上的分叉。一个可执行节点根据添加的构造型的不同可以转换成 3 种 GSPN 结构,分别对应资源主动执行的动作、获取资源的动作和释放资源的动作。

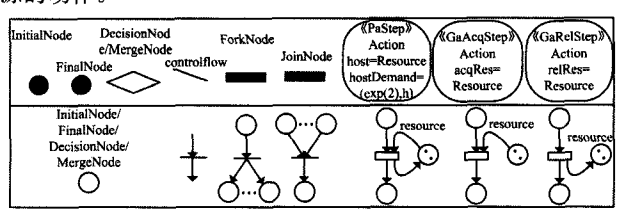

图 6 活动图的转换规则

图 7 和图 8 分别列举了两段具体的转换代码,用来说明 ATL 转换过程。图 7 给出了活动图中控制流的转换规则代码。变量 *counter* 是一个由 ATL helper 定义的全局变量,用来对转换成的 GSPN 元素进行计数和命名。ATL helper 还

可以定义一些辅助函数，方便在转换时调用，例如第 14 行中的 *valueOfWeight*，如果该控制流从 *decision* 节点发出且添加构造型 *PaStep*，该函数就可以返回构造型中的属性 *prob* 值，否则，返回默认值 1。

```
1. rule ControlFlow{
2.     from
3.     s:UML!ControlFlow
4.     to
5.     PT1:PetriNets! PlaceToTransition (
6.         name <-…,
7.         multiplicity <-1
8.     ),
9.     T1:PetriNets! Transition (
10.        name <-…,
11.        incoming <- PT1,
12.        type <- 'Immediate',
13.        priority <-1,
14.        weight <-s. valueOfEdgeWeight,
15.        firingrate <- 0.0
16.    ),
17.    TP1:PetriNets! TransitionToPlace(
18.        name <-…,
19.        source<-T1,
20.        multiplicity<-1
21.    )
22. }
```

图 7　控制流的转换规则代码

```
1. rule ForkNode{
2.     from
3.     s:UML! ForkNode
4.     to
5.     P1:PetriNets! Place (
6.         name <- …,
7.         incoming<- thisModule. resolveTemp
8.         (s. incoming. first(),'TP1')
9.     ),
10.    PT1:PetriNets! PlaceToTransition(
11.        name<-…,
12.        source<-P1,
13.        multiplicity<-1
14.    ),
15.    T1:PetriNets! Transition (
16.        name<-…,
17.        incoming<-PT1,
18.        type<-'Immediate',
19.        priority<-1,
20.        weight<-1,
21.        firingrate<-0
22.    ),
23.    TPs:distinct PetriNets! TransitionToPlace
24.        foreach(e in s. outgoing) (
25.        name<-…,
26.        source<-T1,
27.        target<-Ps,
28.        multiplicity<-1
29.    ),
30.    Ps:distinct PetriNets!Place
31.        foreach(e in s. outgoing) (
32.        name<-…,
33.        incoming<-TPs,
34.        outgoing<-thisModule. resolveTemp
35.        (e,'t1')
36.    )
37. }
```

图 8　fork 节点的转换规则代码

活动图中 fork 节点的转换规则代码如图 8 所示。库所 P1 需要与 fork 节点的输入控制流转换成的 GSPN 结构相连，这是通过函数 resolveTemp(第 7 行)实现的，其作用是将不同模块中的元素关联在一起，具体地，P1 的输入弧被设为图 7 中控制流转换规则代码中的弧 TP1。对于具有多条分支的 fork 结构，变迁 T1 的输出弧和输出库所要生成相应数目的结构，这是通过 distinct for each()结构(第 22,28 行)来实现的，每一个分支都转换成一个输出弧和库所相连的结构。

模型转换过程中，UML 模型中的所有必要的元素、属性以及关联关系被映射到 GSPN 模型中的库所、变迁和弧上，转换而来的 GSPN 模型可以直接用来进行性能分析，但往往会存在冗余结构，冗余结构会产生冗余状态，增大状态空间，从而加大分析的难度，以至于不能够得出分析结果，这就是所谓的"状态空间爆炸"的问题。因此，一方面需要在转换过程中尽量减少引入冗余结构，另一方面可以通过压缩的方法来消除模型中存在的冗余结构。在转换过程中，为了使得转换规则更加规范，可能人为地引入一些冗余的结构，比如由用例图中的关联弧以及活动图中的控制流转换而成的结构，它们只是起到连接的作用，可以将其消除掉。由于模型中存在着很多这种冗余结构，将其消除之后可以显著地降低模型的规模。最终得到的 GSPN 模型如图 9 所示。

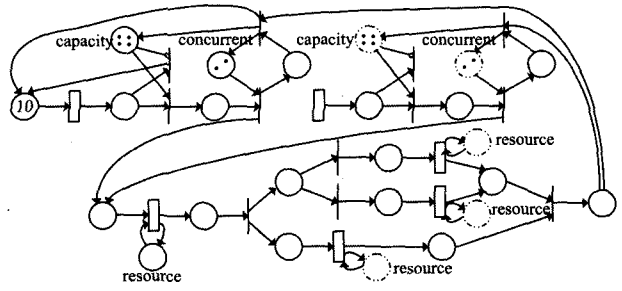

图 9　由 UML 模型(图 2)转换得到的 GSPN 模型

ATL 实现了语义转换，但得到的模型往往不能直接被分析工具所使用，还需要经过语法转换，转换成分析工具所支持的格式。这种文本转换可由 XStream 实现，其过程是首先分别根据不同的模型文件格式建立相对应的源类和目标类，然后读取 ATL 技术生成的 GSPN 模型文件，建立源类的对象，最后通过 XStream 实现源类和目标类的转换，从而实现源类对象到目标类对象的转换。通过语法转换可以生成多种格式的模型文件，便于采用不同的分析工具分析模型，如 SPNP[17] 和 WebSPN[18]，不同的工具具有不同的功能，从而能够对模型进行更好的分析。

4 性能分析

基于GSPN的性能分析方法主要包括3个步骤：首先给出一个GSPN模型，然后构造与该GSPN模型同构的马尔可夫链，最后基于马尔可夫链的稳定状态概率计算性能指标，从而可以根据这些性能指标评价系统的性能。

本文考虑的性能指标主要与资源有关，包括利用率、吞吐量、平均等待请求的数目以及响应时间。

利用率是资源被使用的概率，其公式为：

$$\sum_i \pi_i \times 1(m_i(P_{res}) < resMult)$$

其中，π_i 表示标识 m_i 的稳定状态概率；$1(\cdot)$ 是指示函数，当条件为真时，返回1，否则，返回0；$m_i(\cdot)$ 表示某个库所在标识 m_i 下token的数目；P_{res} 表示代表资源的库所；$resMult$ 表示资源的数目。

吞吐量是资源在单位时间内执行动作的次数，计算公式为：

$$\sum_i \pi_i \times 1(T_{res} \text{ is enabled}) \times rate(T_{res})$$

其中，T_{res} 是一个时间变迁，代表资源 res 执行的动作；$rate(T_{res})$ 表示变迁 T_{res} 的执行速率。

平均等待请求的数目是资源等待队列的平均长度，计算公式为：

$$\sum_i \pi_i \times m_i(P_{wait\_res})$$

其中，P_{wait_res} 是一个库所，代表资源等待队列，即资源执行动作的时间变迁的输入库所。

此外，还有一些与系统有关的指标，比如丢失率和并发用户数等。丢失率是系统内用户的数目达到容量限制而导致新到达的用户被拒绝进入系统的概率，即图5中变迁T2的使能概率；而并发用户数指的是系统中处在运行状态的用户的数目，与图5中库所concurrent的token数目有关。

可以看出，基于GSPN的性能分析方法可以设置不同的性能指标，方便考察系统性能的各个方面，通过调节UML模型的结构和参数可以研究系统在不同场景下的性能。

结束语 UML模型一般不能直接进行性能分析，需要利用模型转换的方法将其转换成其他分析模型。本文通过利用Eclipse平台上的Papyrus建立3种类型的UML模型(用例图、部署图和活动图)来对系统进行建模，利用MARTE规范添加性能相关的信息，并利用ATL实现了从UML模型到广义随机Petri网(GSPN)模型的自动转换，使用XStream将自动转换得到的GSPN模型再转换成分析工具所支持的格式；同时给出了一系列性能指标的计算方法，如利用率、吞吐量、平均等待请求的数目以及响应时间等，为设计和开发人员提供了多方面的性能相关信息，从而能够方便找到并解决性能瓶颈问题，使得性能得到优化，这是对模型驱动工程的一次成功的运用。

未来的工作主要包括两个方面：1)对UML模型进行扩展，使之可以对更多复杂的系统进行建模；2)增加对其他后端分析模型的支持，比如随机良构有色网模型，进而可以得到更为紧凑的模型。

参 考 文 献

[1] OMG specifications[OL]. http://www.omg.org/spec

[2] Woodside M, Petriu D C, Merseguer J, et al. Transformation challenges: from software models to performance models[J]. Software & Systems Modeling, 2014, 13(4): 1529-1552

[3] Brosig F, Meier P, Becker S, et al. Quantitative evaluation of model-driven performance analysis and simulation of component-based architectures[J]. IEEE Transactions on Software Engineering, 2015, 41(2): 157-175

[4] King P J B, Pooley R. Derivation of Petri net performance models from UML specifications of communications software[M]// International Conference on Computer PERFORMANCE Evaluation: Modelling Techniques and TOOLS. Springer-Verlag, 2000

[5] Merseguer J, Campos J, Bernardi S, et al. A compositional semantics for UML state machines aimed at performance evaluation[C]// Proc. of the Sixth International Workshop on Discrete Event Systems. IEEE, 2002: 295-302

[6] Bernardi S, Donatelli S, Merseguer J. From UML sequence diagrams and statecharts to analysable Petri net models[C]// Proc. of the 3rd International Workshop on Software and Performance. ACM, 2002: 35-45

[7] Storrle H. Semantics of control-flow in UML 2.0 activities[C]// 2004 IEEE Symposium on Visual Languages and Human Centric Computing. IEEE, 2004: 235-242

[8] Yang N, Yu H, Sun H, et al. Mapping UML activity diagrams to analyzable Petri net models[C]// 2010 10th International Conference on Quality Software (QSIC). IEEE, 2010: 369-372

[9] Heuer A, Stricker V, Budnik C J, et al. Defining variability in activity diagrams and Petri nets[J]. Science of Computer Programming, 2013, 78(12): 2414-2432

[10] Eshuis R, Wieringa R. Comparing Petri net and activity diagram variants for workflow modelling—a quest for reactive Petri nets [M]. Petri Net Technology for Communication-Based Systems. Springer Berlin Heidelberg, 2003: 321-351

[11] López-Grao J, Merseguer J, Campos J. From UML activity diagrams to Stochastic Petri nets: application to software performance engineering[J]. ACM SIGSOFT Software Engineering Notes, ACM, 2004, 29(1): 25-36

[12] Campos J, Merseguer J. On the integration of UML and Petri nets in software development[M]// Petri Nets and Other Models of Concurrency (ICATPN 2006). Springer Berlin Heidelberg, 2006: 19-36

[13] Distefano S, Scarpa M, Puliafito A. From UML to Petri nets: the PCM-based methodology[J]. IEEE Transactions on Software Engineering, 2011, 37(1): 65-79

[14] Papyrus v0.10.1[OL]. http://www.eclipse.org/papyrus

[15] ATL v3.4.0[OL]. http://www.eclipse.org/atl

[16] XStream v1.4.6[OL]. http://xstream.codehaus.org

[17] Ciardo G, Muppala J, Trivedi T. SPNP: stochastic Petri net package[C]// Proc. of the 3rd International Workshop on Petri Nets and Performance Models (PNPM89). IEEE, 1989: 142-151

[18] Bobbio A, Puliafito A, Scarpa M, et al. WebSPN: A WEB-accessible Petri net tool[C]// Proceedings of the Conference on Web-based Modeling & Simulation. 1998