# 基于规则的软件体系结构层性能优化模型

杜 欣[1] 汪春燕[1] 倪友聪[1] 叶 鹏[2] 肖如良[1]
(福建师范大学软件学院 福州 350108)[1] (武汉纺织大学数学与计算机学院 武汉 430073)[2]

**摘 要** 针对大多数基于规则的软件体系结构层性能优化方法在优化过程中未充分考虑规则的使用次数和使用顺序而导致搜索性能改进空间受限,难以获取最优性能改进方案的问题,设计一种规则序列执行框架,并进一步将软件体系结构层性能优化抽象为求解最优规则序列的数学模型,以精确刻画规则的使用次数、使用顺序与最优性能改进方案之间的数学关系,为搜索更大的性能改进空间、提高优化质量提供支持。

**关键词** 性能分析,性能优化,规则,软件体系结构
**中图法分类号** TP311 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2015.10.038

## Rule-based Performance Optimization Model at Software Architecture Level

DU Xin[1] WANG Chun-yan[1] NI You-cong[1] YE Peng[2] XIAO Ru-liang[1]
(Faculty of Software, Fujian Normal University, Fuzhou 350108, China)[1]
(College of Mathematics and Computer, Wuhan Textile University, Wuhan 430073, China)[2]

**Abstract** The use number and order of rules in the performance improvement process have not been fully considered in the most of rule-based approaches to performance improvement at software architecture level. As a result, the search space for performance improvement is limited so that the optimal solution for performance improvement is hard to find out. Aiming at the problem, this paper firstly designed a rule sequence execution framework (RSEF). Furthermore, performance improvement at software architecture level was abstracted into the mathematical model called RPOM for solving the optimal rule sequence. In the RPOM model, the mathematical relation between the usage of rules and optimal solution for performance improvement is precisely characterized. The result of this paper will support the rule-based performance improvement approaches in searching the larger space for performance improvement and improving the quality of optimization.

**Keywords** Performance analysis, Performance optimization, Rule, Software architecture

## 1 引言

软件的性能是衡量软件系统质量的一个重要属性,性能的优劣已经成为系统成败的关键因素。在软件体系结构(Software Architecture,SA)设计阶段进行性能优化,可以尽早发现资源使用率过高、响应时间过长和吞吐量过小等性能问题,并通过相应的设计改进缓解或消除这些问题,以获得满足性能需求的SA设计方案,进而可在软件生命周期的早期达到性能优化的目的。基于SA进行性能优化,不仅可以显著减少软件性能优化的时间,而且能够有效降低优化的成本。

SA层性能优化一直是软件工程学术界和工业界高度关注的主题,人们已提出了分层排队网(LQN)、随机Petri网、随机进程代数和马尔可夫链等性能模型及分析工具[1-3],为SA层的性能评估、预测和优化提供了有力支持。一些基于SA的性能评估、预测和优化方法不断地涌现出来。Koziolek等人对基于组件系统SA的性能评估方法进行了较为全面的总结和分析[4]。Gomaa等人将SA风格对系统性能的影响整合到性能预测模型中,提出了一种考虑SA风格的性能预测方法[5]。Brosig等人提出了一种面向服务SA的在线性能预测方法[6]。Christoph等人针对基于事件通信风格的SA,提出了一种性能建模及预测方法[7]。Koziolek等人提出了一种融合设计策略的软件性能自动优化方法[8],该方法将SA设计策略应用到演化优化过程中。

经过多年研究和实践,在考虑设计SA风格和策略的基础上,通过分析SA的静态结构、动态行为和部署方案,人们已能够对最终软件系统的性能给出较为准确的评估和预测,并已积累了一些普适性的SA层性能改进知识。这些知识已被系统地归纳和总结,并运用自然语言描述的性能反模式[9,10]进行了相应表示。为了进一步提高SA层性能优化的自动化程度,近年来已涌现出一些基于规则的SA层性能优化方法。Xu等人提出了一种基于LQN模型[11]的SA层性能优化方法[12]。该方法运用Jess框架定义了一组性能的诊

到稿日期:2014-10-05 返修日期:2015-01-12 本文受国家自然科学基金(61305079,61370078),福建省自然科学基金(2015J01235),武汉大学软件工程国家重点实验室开放基金(SKLSE2014-10-02),福建省教育厅JK类项目(JK2015006)资助。

杜 欣(1979—),女,博士,副教授,CCF高级会员,主要研究方向为基于搜索的软件工程、智能计算等,E-mail:xindu79@126.com;汪春燕(1989—),女,硕士生,主要研究方向为基于搜索的软件设计;倪友聪(1976—),男,博士,副教授,CCF会员,主要研究方向为软件体系结构、基于搜索的软件设计等,E-mail:youcongni@foxmail.com;叶 鹏(1976—),男,博士,讲师,主要研究方向为软件体系结构;肖如良(1966—),博士,教授,主要研究方向为设计模式与软件体系结构。

断和改进规则，并借助所设计的优化算法使用这些规则在设计空间和配置空间进行搜索，以发现瓶颈和长路径等软件性能问题，并自动执行相应的改进方案。Mcgregor 等人开发了用于分析评估 SA 可修改性和性能的 ArchE 框架[13]。该框架提供了基于规则的步进式搜索方式，当发现可修改性和性能问题后，可给出预定义的改进方案。Bondarev 等人提出了 DeepCompass 框架[14]。该框架能够基于 SA 探测和发现嵌入式系统中存在的性能问题，并给出成本相对较低的改进方案。Cortellessa 和 Trubiani 等人基于传统的排队网模型和一阶谓词描述了一组性能反模式[15-18]，并基于这些谓词和 SA 的 XML 表示，构建了侦测性能反模式的引擎。该引擎可以自动发现性能问题并给出相应的改进方案。

目前基于规则的方法已能以机器可处理的规则形式显式、精确地描述性能反模式，并提供相应的规则执行引擎，可在优化过程中自动地应用这些规则发现 SA 层性能问题并通过预定义的方案进行性能改进。然而，这些方法大多未充分考虑规则组合使用场景下各规则的使用次数和使用顺序不确定性的问题，因此只能搜索相对较小的性能改进空间，往往难以获得最优的性能改进方案。例如，针对配置不当而引发资源瓶颈的性能问题，Xu 等人定义了变更资源使用数目和重新配置资源两条规则，并总是按先变更资源使用数目再重新配置资源的固定顺序来缓解资源瓶颈问题[12]。实际在组合使用这两条规则时，每条规则均可使用多次，甚至不使用某条规则，在多次使用这两条规则的情形下还存在使用顺序的问题。因而，在优化过程中只有充分考虑了多条规则使用的各种组合情况，才有可能获取相对较优的改进方案。然而，一个不容忽视的事实是 SA 层性能改进规则的组合使用空间往往较为庞大。例如，Cortellessa 和 Trubiani 等人针对 12 种 SA 层性能反模式[9]，给出了对应的 12 条性能改进规则[18]。即使在不考虑规则可以重复使用的情况下，由这 12 条规则不同组合的情况而构成的改进空间也将有高达 $12^{12}$（近 9000 万亿）个改进方案。在如此庞大的 SA 层性能改进空间中，如何建立规则的使用次数、使用顺序与最优性能改进方案之间的关系，进而为找出最优性能改进方案提供支持，仍是一个亟待解决的问题。针对上述问题，本文设计一种 SA 层性能改进规则序列的执行框架 RSEF；并进一步对基于规则的 SA 层性能优化问题进行抽象，提出一种 SA 层性能优化模型 RPOM。该模型可精确描述优化过程中规则的使用次数、使用顺序与最优性能改进方案之间的数学关系，为搜索更大的性能改进空间、提高优化质量提供支持。

本文第 2 节对已有的 SA 层性能改进规则执行引擎进行了抽象说明；第 3 节详细阐述了 SA 层性能改进规则序列的执行框架；第 4 节提出基于规则的 SA 层性能优化模型；最后总结全文并给出未来的研究方向。

## 2 SA 层性能改进规则的执行引擎

基于 SA 的性能评估技术是构建 SA 层性能改进规则执行引擎的基础。图 1 显示了基于 SA 的性能评估引擎[7,13]的总体结构。待建系统的 SA 经过抽取引擎的抽取可生成某种性能模型的一个实例，再通过该种性能模型的解析器解析，能输出对应的性能评估报告，最后借助性能指标读取器可获取待建系统的各项性能指标，如响应时间、吞吐量和资源使用率等（本文仅讨论响应时间，下同）。

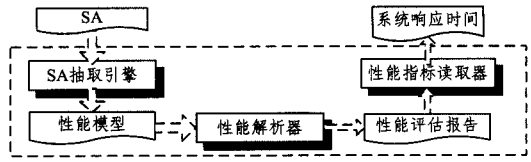


图 1 基于 SA 的性能评估引擎的总体结构

在基于 SA 的性能评估引擎基础上，根据 SA 层性能改进规则的机器内部表示[12,18]，基于规则的性能优化方法已提供了相应的性能改进规则执行引擎。图 2 显示了基于 SA 的性能改进规则执行引擎的总体结构。SA 变换引擎根据性能改进规则可对 SA 进行变换，生成改进后 SA，借助性能评估引擎，可获取改进前后的系统性能指标，如响应时间。

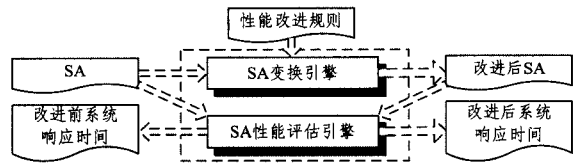


图 2 基于 SA 的性能改进规则执行引擎的总体结构

虽然规则执行引擎已能较好地借助单条规则的执行实现 SA 层性能改进知识的自动化应用，但在性能优化过程中仍需充分考虑多条规则不同组合使用的情形。

## 3 SA 层性能改进规则序列的执行框架 RSEF

基于 SA 层性能改进规则执行引擎，并充分考虑性能优化过程中规则的使用次数和使用顺序，设计了一种 SA 层性能改进规则序列的执行框架 RSEF。图 3 给出了 RSEF 框架的总体结构。

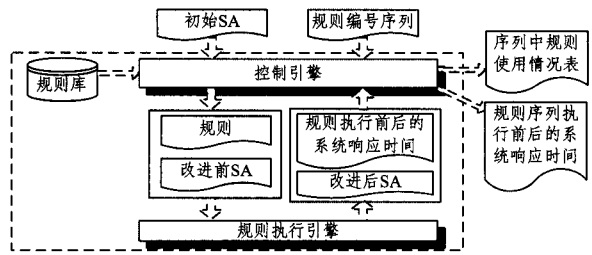


图 3 SA 层性能改进规则序列的执行框架 RSEF 的总体结构

下面给出 RSEF 框架涉及的主要数据结构及规则序列执行过程的定义。

### 3.1 数据结构定义

①规则：机器可处理的 SA 层性能改进规则。

②规则号序列：为了灵活应用 SA 层性能改进规则进行性能优化，给每个规则指定一个唯一的整数编号，并通过规则号序列来指示需要顺序使用的一组规则。

③规则库：集中存储统一编号后的 SA 层性能改进规则，可按照规则号从规则库提取与之对应的规则。

④初始 SA：需要进行性能改进的最初软件体系结构。

⑤改进前 SA 和改进后 SA：分别表示一条 SA 层性能改进规则执行之前的软件体系结构，以及该规则应用到执行前软件体系结构后获取的软件体系结构。

⑥规则执行前后的系统响应时间：分别表示根据一条规则执行前后对应软件体系结构而计算出的系统响应时间。

⑦规则序列执行前后的系统响应时间：分别表示根据一个规则序列执行前后对应软件体系结构而计算出的系统响应时间。

⑧序列中规则使用情况表：序列中规则使用情况表 T_RuleUseInSeq 的设计如表 1 所列。rulNum 和 loc 两个字段构成了 T_RuleUseInSeq 表的主码。

表 1　T_RuleUseInSeq 表的设计

| 序号 | 字段名称 | 含义 |
|---|---|---|
| 1 | rulNum | 规则编号 |
| 2 | loc | 规则编号在序列中的位置 |
| 3 | isImp | 根据 loc 位置上的编号为 rulNum 的规则执行前后的系统响应时间，判定该规则的应用是否有性能改进效果。有改进效果时为 1，否则为 0 |

### 3.2　规则序列的执行过程定义

在预定义的数据结构基础上，规则序列的执行过程由控制引擎和规则执行引擎共同协作完成，具体由以下算法定义。

输入：规则号序列 X，初始软件体系结构 $SA_0$

输出：X 对应规则序列执行前的系统响应时间 BRT 和执行后系统响应时间 ART、序列中规则使用表 T_RuleUseInSeq

S1. 将待改进软件体系结构 SA、当前位置 i、T_RuleUseInSeq 表、BRT 和 ART 分别赋值为初始软件体系结构 $SA_0$、1、空、−1 和−1。

S2. 判断 i 是否大于 X 的长度，是则转 S6；否则转 S3。

S3. 取 X 的第 i 个元素 $x_i$，并根据编号 $x_i$ 从规则库取出对应的规则，并与待改进软件体系结构 SA 一并交由规则执行引擎执行。

S4. 根据规则执行引擎返回的执行前系统响应时间 $BRT_i$ 和执行后系统响应时间 $ART_i$，计算 $BRT_i$ 与 $ART_i$ 的差值 $\Delta_{RT}$。

S41. 若 BRT 为−1，则把 $BRT_i$ 赋值给 BRT；

S42. 将 $ART_i$ 赋值给 ART；

S43. 若 $\Delta_{RT}>0$，将记录$(x_i,i,1)$插入 T_RuleUseInSeq 表中，否则将记录$(x_i,i,0)$插入 T_RuleUseInSeq 表中；

S44. 若 $\Delta_{RT}>0$，则将规则执行引擎返回的改进后软件体系结构赋值给 SA，使其成为下一条规则执行时的待改进软件体系结构。

S5. 令 i=i+1，并转 S3。

S6. 输出 BRT、ART 和 T_RuleUseInSeq 表，算法结束。

## 4　基于规则的 SA 层性能优化模型 RPOM

基于 RSEF 规则序列执行框架，可将 SA 层性能优化抽象为求解最优规则序列的数学模型 RPOM。下面对 RPOM 模型进行详细阐述。

从 1 至 $n$ 依次对 $n$ 条性能改进规则进行编号。定义规则号序列 $X=\langle x_1,x_2,\cdots,x_k,\cdots,x_l\rangle$ 表示性能改进方案，并用 $u_i$ 表示规则号 $i$ 在 $X$ 中最多可出现的次数。$X$ 的长度 $l$、$X$ 中每个元素 $x_k$ 的取值范围、规则编号 $i$ 在 $X$ 中的出现次数 $h_i(X)$ 分别由式(1)—式(3)定义：

$$l\leqslant\sum_{i=1}^{n}u_i \tag{1}$$

$$x_k\in \mathrm{N}\wedge 1\leqslant x_k\leqslant n\wedge 1\leqslant k\leqslant l,\ \mathrm{N}\text{ 为自然数集} \tag{2}$$

$$h_i(X)\leqslant u_i,\ 1\leqslant i\leqslant n \tag{3}$$

定义函数 $imp(q,SA)$ 用于判定编号为 $q$ 的规则应用到待改进软件体系结构 $SA$ 后，是否有性能改进。其定义如式(4)所示：

$$imp(q,SA)=\begin{cases}0, & \text{当编号为 } q \text{ 的规则应用于 } SA \text{ 后}\\ & \text{没有性能改进}\\ 1, & \text{否则}\end{cases} \tag{4}$$

$$q\in \mathrm{N}\wedge 1\leqslant q\leqslant n$$

定义变换函数 $t(X,SA)$ 表示依次应用性能改进方案 $X$ 所指示的一组规则到软件体系结构 $SA$ 上，最后得到的改进后软件体系结构。函数 $t$ 由式(5)和式(6)定义。其中，序列 $Y=\langle 1,2,\cdots,n\rangle$，其第 $i$ 个元素用 $y_i$ 表示；$Z=X/x_1$ 和 $W=\langle x_1\rangle$ 分别表示去除 $X$ 中第一个元素和仅包含 $X$ 中第一个元素的序列。

$$t(Y_i,SA)=\begin{cases}SA, & imp(y_i,SA)=0\\ SA_i, & imp(y_i,SA)=1\end{cases} \tag{5}$$

$$Y_i=\langle y_i\rangle\wedge 1\leqslant i\leqslant n$$

$$t(X,SA)=t(Z,t(W,SA)) \tag{6}$$

定义函数 $r(SA)$ 表示根据软件体系结构 $SA$ 求解系统响应时间。

定义函数 $g(X)$ 表示性能优化对应的目标函数，其定义如式(7)所示。其中：$SA_0$ 和 $h_i(X)$ 分别为初始软件体系结构、规则编号 $i$ 的出现次数，序列 $V_i=\langle x_1,\cdots,x_{i-1}\rangle$。$g(X)$ 的两个乘积项分别表示性能改进幅度、$X$ 中有改进效果规则的数目占规则总数的比例，$g(X)$ 的值越大表明性能改进方案 $X$ 越优。

$$g(X)=(r(SA_0)-r(t(X,SA_0)))\times\frac{imp(x_1,SA_0)+\sum_{i=2}^{l}imp(x_i,t(V_i,SA_0))}{\sum_{i=1}^{n}h_i(X)} \tag{7}$$

基于规则的 SA 层性能优化问题可建模为：在满足式(1)—式(3)的条件下，求解 $X$ 使得 $g(X)$ 最大。

**结束语**　基于已有的 SA 层性能改进规则执行引擎，并充分考虑性能优化过程中规则的使用次数和使用顺序，设计了一种 SA 层性能改进规则序列的执行框架 RSEF。具体定义了 RSEF 框架涉及的主要数据结构及规则序列执行过程。在此基础上，将 SA 层性能优化抽象为求解最优规则序列的数学模型 RPOM。该模型精确地刻画了规则的使用次数、使用顺序与最优性能改进方案之间的关系，进而为增大搜索空间、提高优化质量提供支持。文中定义的 RSEF 框架和提出的 RPOM 模型具有较好的通用性，对于改进已有基于规则的 SA 层性能优化方法具有一定指导意义。未来将设计高效的演化算法对 SA 层最优性能改进方案进行求解，以提出一种基于规则的 SA 层性能演化优化方法。

## 参 考 文 献


[1] Bernardo M, Hillston J. Formal Methods for Performance Evaluation[C]// 7th International School on Formal Methods for the Design of Computer, Communication and Software Systems. Bertinoro, Italy: Springer, 2007: 1-37

[2] Marco D, Antinisca V C, Inverardi P. Model-based software performance analysis [M]. Berlin: Springer, 2011

[3] 李传煌，王伟明，施银燕. 一种 UML 软件架构性能预测方法及其自动化研究[J]. 软件学报，2013，24(7)：1512-1528
Li Chuan-huang, Wang Wei-ming, Shi Yin-yan. Performance Prediction Method for UML Software Architecture and its Automation[J]. Journal of Software, 2013, 24(7): 1512-1528

[4] Becker S, Koziolek H, Reussner R. The Palladio component model for model-driven performance prediction [J]. Journal of System and Software, 2009, 82(1): 3-22

[5] Gomaa H, Menascé D A. Design and performance modeling of component interconnection patterns for distributed software architectures[C]// Proceedings of the Second International Workshop on Software and Performance. Ontario, Canada: ACM, 2000: 117-126

[6] Brosig F, Huber N, Kounev S. Architecture-level software performance abstractions for online performance prediction [J]. Science of computer programming, 2014, 90(B): 71-92

[7] Rathfelder C, Klatt B, Sachs K. Modeling event-based communication in component-based software architectures for performance predictions [J]. Software & Systems Modeling, 2014, 13 (4): 1291-1317

[8] Koziolek A, Koziolek H, Reussner R. Peropteryx: automated application of tactics in multi-objective software architecture optimization[C]// Proceedings of the joint ACM SIGSOFT Conference — QoSA and ACM SIGSOFT Symposium — ISARCS on Quality of Software Architectures. Boulder, Colorado, USA: ACM, 2011: 33-42

[9] Smith C U, Williams L G. Software performance antipatterns [C]// Proceedings of the 2nd International Workshop on Software and Performance. New York, NY, USA: ACM, 2000: 127-136

[10] Smith C U, Williams L G. Performance solutions: a practical guide to creating responsive, scalable software [M]. Addison-Wesley, 2002

[11] Tribastone M. A fluid model for layered queueing networks[J]. IEEE Transactions on Software Engineering, 2013, 39(6): 744-756

[12] Xu J. Rule-based automatic software performance diagnosis and improvement [J]. Performance Evaluation, 2012, 69(11): 525-550

[13] Mcgregor J D, Bachmann F, Bass L, et al. Using arche in the classroom: One experience (CMU/SEI-2007-TN-001) [R]. Pittsburgh: Software Engineering Institute, Carnegie Mellon University, 2007

[14] Bondarev E, Chaudron M R V, de Kock E A. Exploring performance trade-offs of a JPEG decoder using the DeepCompass framework[C]// Proceedings of the 6th international workshop on software and performance, 2007. Buenos Aires, Argentina: ACM, 2007: 153-163

[15] Cortellessa V, Frittella L. A framework for automated generation of architectural feedback from software performance analysis[C]// Proceedings of Fourth European Performance Engineering Workshop (EPEW 2007). Berlin: Springer, 2007: 171-185

[16] Cortellessa V, Martens A, Reussner R, et al. A process to effectively identify "guilty" performance antipatterns [M]. Fundamental approaches to software engineering. Berlin Heidelberg: Springer, 2010: 368-382

[17] Trubiani C, Koziolek A. Detection and solution of software performance antipatterns in palladio architectural models[C]// Proceedings of the 2nd ACM/SPEC International Conference on Performance Engineering. New York, NY, USA: ACM, 2011: 19-30

[18] Cortellessa V, Di Marco A, Trubiani C. An approach for modeling and detecting software performance antipatterns based on first-order logics [J]. Software & Systems Modeling, 2014, 13 (1): 391-432

(上接第 183 页)

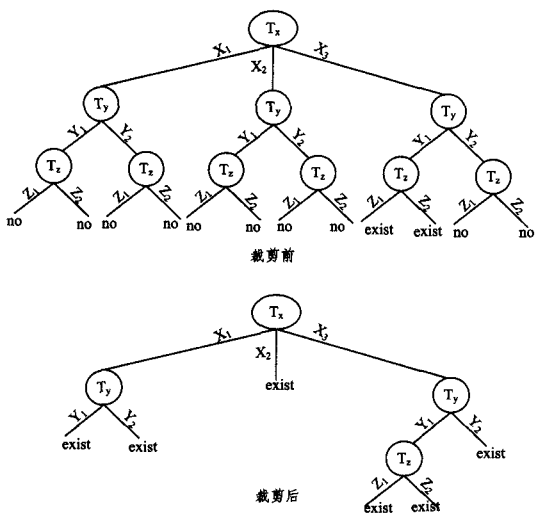


图 8 标记完的判定树

**结束语** 测试任务流模型的结构特性、行为特性和性能方面的分析是很重要且必要的，但是其中或分支的完整性的分析也是很有必要的。本文从分析或分支的完整性入手，给出其完整性定义，提供一种验证或分支完整性的判定树方法。该方法不依赖于具体的模型，适用性好；算法借鉴哈夫曼树的思想，有一定理论基础，易于编程实现。当然，该方法的前提是需要保证每一约束变量自身是完整的，这一点由人保证问题不大。本文的研究只是针对测试任务流模型中的或分支完整性，没有对与分支和循环等控制结构进行分析验证，这些在将来的研究中需要进一步的完善。

## 参 考 文 献

[1] IEEE Standards Coordinating Committee 20. IEEE Std 1671-2010, IEEE Standard for Automatic Test Markup Language (ATML) for Exchanging Automatic Test Equipment and Test Information via XML [S]. USA: IEEE-SA Standards Board, 2010

[2] IEEE Standards Coordinating Committee 20. IEEE Std 1671. 1-2009, IEEE Trial-Use Standard for Automatic Test Markup Language (ATML) for Exchanging Automatic Test Equipment and Test Information via XML: Exchanging Test Descriptions [S]. USA: IEEE-SA Standards Board, 2009

[3] Muth P, Wodtke D, Weissenfels J. Enterprise-Wide workflow management based on state and activity charts[OL]. http://paris. cs. uni-sb. de/public_html/papers/nato-wf: ps

[4] Scholz-Reiter B, Stichel E. Business process modeing[M]Berlin: Springer-Verlag, 1996

[5] 袁崇义. Petri 网原理与应用[M]. 北京：电子工业出版社，2005: 225-258
Yuan Cong-yi. The Theory and Application of Petri Nets[M]. Beijing: Publishing Company of Electric Industry, 2005: 225-258

[6] 罗海滨，范玉顺，吴澄. 工作流合理性验证中的事件平衡分析[J]. 软件学报，2002, 13(8): 1686-1691
Luo Hai-bin, Fan Yu-shun, Wu Cheng. Analysis of event balance in the verification of workflow soundness[J]. Journal of Software, 2002, 13(8): 1686-1691

[7] Ernst W M, Jeremias W. Results on Equivalence, Boundedness, Liveness, and Covering Problems of Bpp-Petri Nets[C]// Application and Theory of Petri Nets and Concurrency 2013. Milan, Italy Jose-Manuel Colom, 2013: 70-90

[8] 庞善臣，蒋昌俊. 一种基于不变量结构分解的工作流性能分析方法[J]. 计算机学报，2010, 33(5): 908-917
Pang Shan-chen, Jiang Chang-jun. Workflow Performance Analysis Based on Invariant Decomposition Algorithm [J]. Chinese Journal of Computer, 2010, 33(5): 908-917