带时间和资源约束的实例化过程模型验证方法①

李伟亨, 谢 淼, 翟 健, 杨秋松

(中国科学院软件研究所 基础软件国家工程研究中心, 北京 100190)
(中国科学院大学, 北京 100049)

摘 要: 软件过程的性能是由软件过程模型和软件过程实例化两方面因素决定, 如果对软件过程进行了不恰当的实例化, 会导致成本超支、进度延期、甚至项目失败. 已有的过程描述法不足以分析实例化过程模型, 由于没有考虑实例化阶段的时间资源约束, 语法结构正确的过程模型并不能保证过程执行的正确性. 提出一种带时间和资源约束的实例化过程模型验证方法, 为目前已有的 s-TRISO/ML 建模语言增加时间和资源约束属性, 然后提出了从 s-TRISO/ML 模型转换成时间自动机的转换方法和实现算法, 利用已有的分析工具 Uppaal 对转换得到的时间自动机的性质进行验证, 得到一个合理的实例化模型, 从而为真实的开发流程提供指导.

关键词: 实例化过程模型; 软件过程; 建模; 验证; 模型检测; 形式化方法

Approach to Verify the Process Instantiation Model with Time and Resource Constraints

LI Wei-Heng, XIE Miao, ZHAI Jian, YANG Qiu-Song

(Laboratory for National Engineering Research Center of Fundamental Software, Institute of Software, Chinese Academy of Sciences, Beijing 100190, China)
(University of Chinese Academy of Sciences, Beijing 100049, China)

Abstract: The performance of software process is related with the software process models and the resource allocations of software processes. If a correct model is impertinently allocated with the limited resource of a software organization, as a result, the performance of the software process may fail to reach the actual requirement, followed by delay, over-cost, and even failure. The existing approach based on process automata isn't fit for the analysis of instantiation model. A model with only correct structure can't ensure a successful enactment, because it lacks schedule information. This paper presents an approach to verify the process instantiation model with time and resource constraints, which is an extension of existing s-TRISO/ML process modeling language. This paper also presents an approach to convert from an s-TRISO/ML model into timed automata and explains converting algorithm. Finally, making use of UPPAAL to carry out the function verification on the converted timed automata, we can get a reasonable instantiation model to provide guidance for the actual process development.

Key words: process instantiation model; software process; modeling verification; model checking; formal method

随着社会的高速发展, 计算机已经成为人类工作和生活不可缺少的助手, 并对整个社会的发展产生了不可替代的影响. 在计算机的进步史上, 计算机软件的发展扮演着一个重要的角色. 计算机软件通常是指计算机系统中的程序、数据及其文档. 计算机软件是计算机的灵魂, 我们每日接触的 PC 终端的操作系统以及手机端的各种应用程序都是不同类型的计算机软件.

软件开发往往具备一定的风险, 若在软件开发中建立过程模型, 可以对软件过程进行有效的指导, 从而减少风险. 所谓软件过程模型就是对软件开发过程的抽象描述. 软件过程建模通过形式化的或半形式化的方法对软件过程进行抽象、表示和分析以增加对软

①基金项目:国家自然科学基金青年基金(61303163)
收稿时间:2014-02-23;收到修改稿时间:2014-03-27

件过程的理解. 软件过程和最终的软件质量紧密相关. 软件过程可以指导实际软件开发活动, 规范软件开发的行为, 使实际的软件开发活动变得可控和可预测.

过程建模和过程执行是所有软件过程支撑环境中必不可少的组成部分. 要保证软件过程的质量, 首先需要保证软件过程模型的无歧义性和正确性. 其次, 需要保证软件过程的性能. 最后, 需要保证软件过程执行与软件过程模型描述的一致性[1].

软件过程实例化[2]是指将软件过程模型要素与软件过程活动属性、软件组织实际要素进行匹配, 使软件组织人员、资源等在软件过程模型的约束下, 开展软件开发活动. 同一个过程模型往往由于过程活动属性、人员能力、成本进度等因素的不同, 从而得到截然不同的过程实例.

在软件过程支撑环境中, 过程模型的语义通常是在运行期由过程引擎来解释的[3]. 传统的过程模型分析发生在运行期, 若在这个期间才发现有限的资源始终无法满足过程执行的实际情况, 那之前建立的过程模型虽然语法结构正确, 但是无法指导实际的过程执行, 浪费了大量的时间和建模成本. 因此, 我们需要在过程执行前对过程模型动态语义进行检验. 而目前已有的过程描述法并没有考虑实例化阶段的时间资源约束, 所以无法分析实例化过程模型.

为了解决如上问题, 我们对带时间和资源约束的实例化过程模型验证方法进行研究, 并在由中国科学院软件所互联网软件技术实验室开发的软件过程建模和分析工具 iTechs Process Studio[1]的基础上, 将之由瑞典 Uppsala 大学的信息技术学院和丹麦 Aalborg 大学的计算科学学院联合开发的 Uppaal[4]工具集成, 具体的研究方法和技术路线如下:

1. 本文首先为目前已有的 s-TRISO/ML (stochasticTRidimensional Integrated SOftware development model/Modelling Language, 基于多元随机 π 演算的图形化软件过程建模语言)建模工具增加与时间和资源相关的约束条件, 让其可以清晰完整地描述软件过程中的相关信息. 例如我们可以为软件过程中每个子活动增加三个基本信息, 最小的执行时间、最大的执行时间和活动涉及到的特定资源.

2. 定义了从 s-TRISO/ML 到时间自动机的转换方法, 并实现转换的算法. 本文已定义七种转换模板, 例如顺序结构、并行结构、选择结构、活动、开始节点、结束节点、活动执行人的翻译转换模板.

3. 基于 Uppaal 的性质验证. UPPAAL 使用分支时序逻辑 CTL 的子集描述需要被验证的性质. 它们是由路径公式和状态公式组成. 状态公式描述了单独的状态, 然而路径公式量化了模型的路径或者轨迹, 路径公式可以用来验证模型的可达性, 安全性和活性. 我们可以利用 Uppaal 完成对软件过程性质的验证, 从而保证得到一个合理的实例化模型, 为真实的开发流程提供指导.

本文的组织结构如下: 第 1 节主要介绍过程模型描述和验证的相关工作; 第 2 节主要介绍 Uppaal、s-TRISO/ML 的背景资料; 第 3 节主要介绍从 s-TRISO/ML 到时间自动机的转换方法和实现算法, 并定义了每种结构的转换思路; 第 4 节主要介绍了利用 Uppaal 验证工具对第 3 节中转换得到的时间自动机的可达性、安全性和活性三种性质进行验证, 并举出了一个验证实例. 最后, 总结全文.

1 相关工作

软件过程研究起源于 20 世纪六七十年代, 卡耐基梅隆大学将软件过程定义为—人们开发与维护软件产品和相关产品所采用的一组互相关联的活动、方法、实践和转化: 它包括工具、方法、材料和人[5]. 一般认为软件过程包含了一组相互关联的基本要素, 例如活动、制品、人员、角色、工具、指导等.

随着“软件产品的质量形成于该软件产品的生产过程[6]等重要思想的出现, 软件过程(Software Process)技术作为软件工程方法的一个分支被正式提出[6], 并逐渐受到了学术界和工业界的广泛关注. 软件过程技术以提高软件开发活动的效率并保证软件产品的质量的目的, 是一门集成了组织、文化、技术、经济、数学等多种学科的综合技术.

在当前研究中, 软件过程建模语言和软件过程建模方法一般被视为相似的概念, 不做明确区分[7,8]. 近二十余年以来, 研究者提出了数十种不同的软件过程建模语言[9-11], 如 SOCCA、MARVEL、ADELE、EPOS(SPELL)、E3、PEACE、SPADE 以及 TRISO/ML 等都是具有代表性的过程建模语言[12-14]. 软件过程建模是指用特定软件过程建模语言建立软件过程的抽象描述[15,16]. 软件过程建模语言主要描述软件过程的要素和要素间的关系[17]. 一种软件过程建模方法所使用

的软件过程建模语言, 决定了该建模方法的严格程度、描述能力、可分析性、可执行性等. 以过程为中心的软件工程环境 (Process-centered Software Engineering Environment, 简称 PSEE)作为软件过程建模语言解释或执行的环境, 也与软件过程建模语言密切相关.

软件过程的性能[1]是由软件过程模型和软件过程实例化两个方面的因素决定的, 即使人们建立了高质量的软件过程模型, 如果对软件过程进行了不恰当的实例化, 那么该软件过程的性能将难以达到客观条件的要求, 进而导致成本超支、进度延期、甚至项目失败.

2 Uppaal和s-TRISO/ML背景介绍

Uppaal 是一个世界领先的实时系统模拟软件, 可对建模成时间自动机网络的实时系统进行形式化规约、确认和验证. 它是由瑞典 Uppsala 大学的信息技术学院和丹麦 Aalborg 大学的计算科学学院于 1995 年联合提出, 它适用于可以被描述为非确定性的并行过程的积的系统. Uppaal 的主要优点是其运作的高效性和使用的方便性, 缺点也较为明显, 例如不易于模拟和调试大规模的工业系统等. Uppaal 一般用于时间由严格要求的系统, 例如实时控制器、通信协议等等.

s-TRISO/ML 是一种形式化的随机软件过程建模语言, 以多元随机 π 演算为基础, 具备图形化和代数表达式两种表达形式, 并且提供从图形化软件过程模型到多元随机 π 演算表达式组的转化规则. 用该语言建立的软件过程模型简明、严格、无歧义, 能够被进一步分析验证, 并且支持随机仿真. 因此, s-TRISO/ML 是一种能够同时满足无歧义性和可验证性要求的软件过程建模语言.

s-TRISO/ML 语言采用面向行为的方式来描述软件过程, 该语言提供了如图 1 所示的图形化符号来描述模型元素. 一个 s-TRISO/ML 软件过程模型由一系列相互交互的实体构成, 其中每一个实体均包含了其内部行为和与其他实体的交互. s-TRISO/ML 使用菱形表示行为之间的顺序、并发和选择关系; s-TRISO/ML 使用椭圆表示过程活动终端节点; s-TRISO/ML 使用椭圆内嵌矩形表示过程活动非终端节点, 过程活动非终端节点能被进一步分解成一组以某种时序关系组合的子活动节点[1].

图 1　s-TRISO/ML 建模语言图形化元素

3 在Uppaal中描述s-TRISO/ML

为了描述实例化阶段的时间资源安排的具体信息, 我们首先为 s-TRISO/ML 建模工具增加了三个约束属性并举出一个实例, 然后定义了从 s-TRISO/ML 到时间自动机的转换方法, 并实现转换的算法.

3.1 属性定义

首先为 s-TRISO/ML 建模工具增加时间、资源的约束. 对于每个 activity, 本文增加了三个属性, 分别为: agent, maxTime, minTime.

下面举一个例子, Activity2 的最小执行时间为 20, 最大执行时间为 50, 需要一个 agent2 能力的人来执行.

Properties

Property	Value
agent	agent2
maxTime	50.0
minTime	20.0
name	Activity2

图 2　带时间约束的示例

3.2 从 s-TRISO/ML 到时间自动机的转换

s-TRISO/ML 具备较为完整的软件过程描述语言的基本属性, 它包含了顺序结构、并行结构、选择结构三个结构属性以及名为活动和通道的实体属性. 本小节针对 s-TRISO/ML 提出的语言框架, 定义了从 s-TRISO/ML 到时间自动机的转换思路, 如下分别介绍顺序结构、并行结构、选择结构、活动、开始节点、结束节点、活动执行人的翻译转换模板.

3.2.1 顺序结构

以 2 个串行活动为例, 当顺序结构收到 Start 信号之后, 顺序结构向 activity1 发出一个 Sequential1Start 信号, 触发第一个分支结构开始执行, 然后等待第一个分支结构结束的信号. 依此类推, 直到收到最后一个分支结构的结束信号时, 此顺序结构才能结束, 并向外发送 end 信号.

3.2.2 并行结构

以 4 个并行活动为例, 并行结构对应两个时间自动机, 分别为 ParallelSplit(并行结构的起始点)和 ParallelJoin(并行结构的汇聚点).

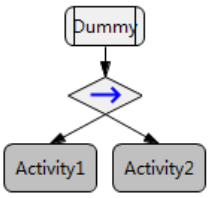

图 3 两个串行活动

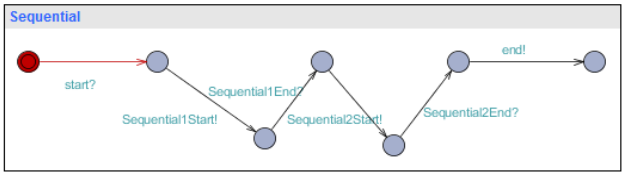

图 4 两个串行活动顺序结构对应的时间自动机

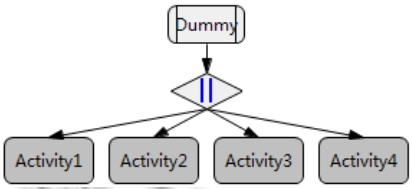

图 5 四个并行活动

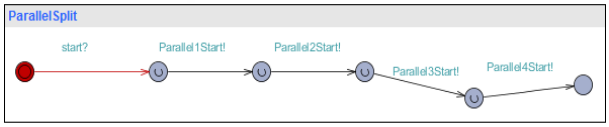

图 6 ParallelSplit 对应的时间自动机

并行结构收到 start 信号之后, 分别向四个分支发送开始信号, 图中表示为(Parallel1Start, Parallel2Start, Parallel3Start, Parallel4Start), 然后所有的子结构开始执行.

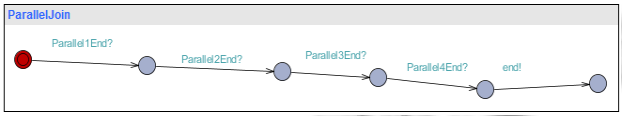

图 7 ParallelJoin 对应的时间自动机

并行结构等待四个分支执行完毕的信号, 当收到全部的信号时, 表示这个并行结构已经执行完毕, 于是向外发送一个结束的信号.

3.2.3 选择结构

选择结构对应两个时间自动机, 分别为 ChoiceSplit(选择结构的起始点)和 ChoiceJoin(选择结构的汇聚点).

3.2.4 活动

每个活动的执行需要达到两个条件, 条件一是收到执行结构的开始信号, 条件二是收到执行人的可用信号. 当两个条件都具备的时候, 我们使用一个 clock 来统计活动实际执行的时间. 当活动完成之后, 我们首先释放执行人的资源, 然后向执行结构发送一个活动结束的信号.

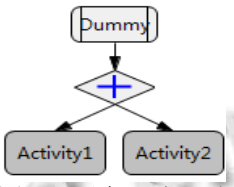

图 8 两个可选活动

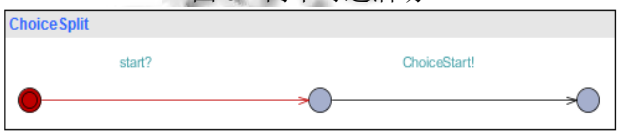

图 9 ChoiceSplit

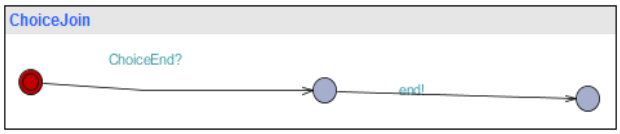

图 10 ChoiceJoin

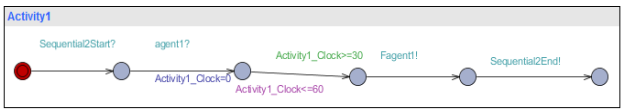

图 11 活动

图 12 开始节点

3.2.5 开始节点

Start 模板只发送一个参数,即一个通道的名称. 这个模板的唯一目的是发送一个过程开始执行的信号.

3.2.6 结束节点

End 模板只接受一个参数,即一个通道的名称. 这个模板的唯一目的是接收一个全部活动完成的信号. 当接收到活动完成的信号时,这意味着整个过程已经执行结束.

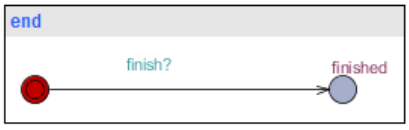

图 13 结束节点

3.2.7 执行人模板

对于每个执行人，我们为其定义一种或若干种执行能力，例如模板中的 person 同时具备三种执行能力，执行人一次只能执行一种活动. 每种执行能力用信号来定义，只有当执行人的相关能力和活动所需的能力相匹配，该活动才能执行.

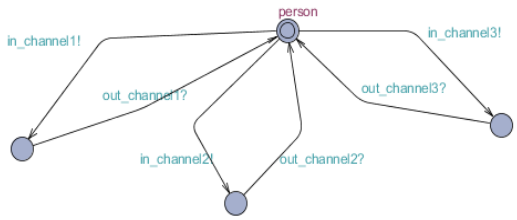

图 14　执行人模板

4　性质验证与实例分析

本节介绍了基于 Uppaal 对可达性、安全性和活性三种性质进行验证所使用的路径公式，并举出了一个验证实例.

4.1 基于 Uppaal 的性质验证

UPPAAL 使用分支时序逻辑 CTL 的子集描述需要被验证的性质. 它们是由路径公式和状态公式组成. 状态公式描述了单独的状态，然而路径公式量化了模型的路径或者轨迹，路径公式可是以下形式：1)使用 E<> p 进行可达性验证的语法验证；若为 true，当且仅当在转换系统中存在一个 S0S1....Sn 序列，使得初始状态为 S0, Sn 为 p; 2)使用 A[] p 或者 E[]p 进行安全性验证，具体指危险情况决不会发生或者某种情况可能决不发生；其中 A[] p 等价于 not E[] not p; 3)使用 A<>p 进行活性验证，满足它表示某种情况将会最终发生；若为 true，当且仅当存在一个 S0S1....Sn 序列，使得 p 在所有状态中有效; 4)也能使用→用于活性验证，满足它表示如果某种情况发生，另外某种情况最终也会发生.

4.2 实例分析

如图 15 所示，我们对一个真实的软件开发过程进行软件过程建模，并初始化如表 1 的数据. 由于资金紧缺，只有 4 个人负责软件开发的全部过程，包括需求分析、软件设计、编码和测试全部阶段. 每个人的能力都不相同，具体初始化为表 2 的数据.

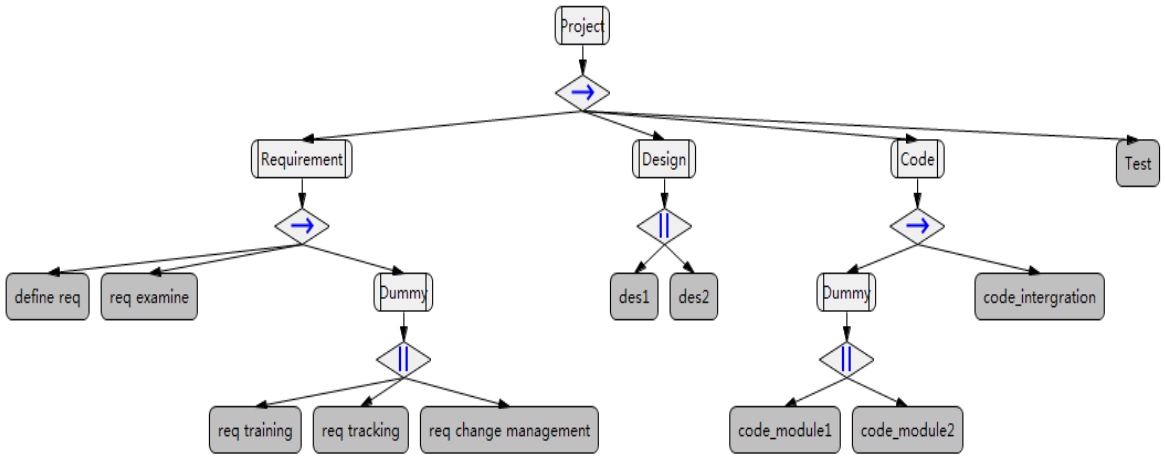

图 15　软件开发过程

表 1　过程的初始化活动信息

活动名	Min Time(h)	Max Time(h)	agent
Define req	30	35	Req
Req examine	10	14	Req
Req training	11	13	Req
Req tracking	12	15	Req
Req change management	20	25	Req
Des1	40	50	Des1
Des2	35	45	Des2
Code_module1	42	48	Code1
Code_module2	55	70	Code2
Code_intergration	70	75	Code3
Test	40	5	test

表 2　过程的初始化活动信息

Person 名	能力 1	能力 2	能力 3
person1	Req	test	
person2	Req	Des1	Des2
person3	Des1	Des2	Code1
person4	Code1	Code2	Code3

针对以上模型和定义, 我们利用本文第三部分阐述的从 s-TRISO/ML 到时间自动机的转换思路自动生成一组时间自动机, 并且使用 Uppaal 作相关性质的验证, 如图 16.

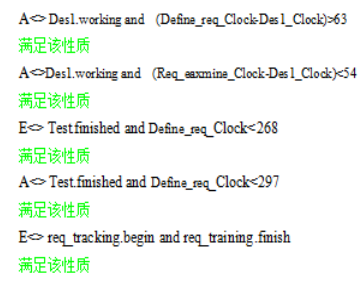

图 16 满足的性质

A<>Des1.working and (Define_req_Clock - Des1_Clock)>63

解释: 在规定的项目计划内, 详细设计 1 总是在需求定义开始后 63 小时之后才能开始.

A<>Des1.working and (Req_eaxmine_Clock - Des1_Clock)<54

解释: 在规定的项目计划内, 详细设计 1 必须在需求检查开始后 54 小时内开始.

E<> Test.finished and Define_req_Clock<268

解释: 在规定的项目计划内, 存在某种情况, 全过程可以在 268 小时内完成.

A<> Test.finished and Define_req_Clock<297

解释: 在规定的项目计划内, 全过程完成的时间不应该超过 297.

E<> req_tracking.begin and req_training.finish

解释: req_tracking 可能在 req_training 之后才开始.

5 结语

本文提出一种带时间和资源约束的实例化过程模型验证方法, 为目前已有的 s-TRISO/ML 建模语言增加时间和资源约束属性, 然后提出了从 s-TRISO/ML 模型转换成时间自动机的转换方法和实现算法, 利用已有的分析工具 Uppaal 对转换得到的时间自动机的可达性、安全性和活性三种性质做验证, 得到一个合理的实例化模型, 从而为真实的开发流程提供指导.

当然本系统还存在一些不足之处, 例如暂时能验证的时间性质比较简单, 验证的算法还可以继续优化等等. 目前 Uppaal 支持的验证性质存在一定的局限性; 若对 Uppaal 工具进行扩展, 那将能更好地利用本方法对实际的软件开发过程进行指导.

参考文献

1 翟健,杨秋松,肖俊超.一种形式化的组件化软件过程建模方法.软件学报,2011,12(1):1–16.
2 潘崇明.基于 Web 的软件过程引擎 WSPE 的设计与实现[硕士学位论文].长沙:湖南大学,2004.
3 董广智,柳军飞,齐璇.一种反应式 SPM 及其动态语义 XYZ 表示.软件学报,2005,16(11):1876–1885.
4 Bengtsson J, Larsen K, Larsson F, Pettersson P, Wang Y. UPPAAL—aToolSuite for Automatic Veilfication of Real-Time Systems. HybridSystems, 1995. 232–243.
5 Paulk M, Curtis B, Chrissis M, et al. Capability maturity model for software, version 1.1. Software IEEE, 1993, 10(4): 18–27.
6 Osterweil L. Software processes are software too. InICSE' 87: Proc. of the 9th international conference on Software Engi-neering. Los Alamitos, CA, USA. IEEE Computer Society Press. 1987. 2–13.
7 Montangero C. The software process: Principles, Methodology, and Technology. Proc. of the Software Process: Principles, Meghodology, Thechnology. Berlin: Springer-Verlag, 1999: 1–13.
8 Podnar I, Mikac B, Caric A. SDL based approach to software process modeling. Proc. of the 7th European Workshop on Software Process Technology. London, UK: Springer-Verlag, 2000: 190–202.
9 Zamli K, Isa N. A survey and analysis of process modeling languages. Malaysian Journal of Computer Science, 2004, 17(2):68–89.
10 Atkinson D, Weeks D, Noll J. The design of evolutionary process modeling languages. Proc. of the Software Engineering Conf on 2004 11th Asia-Pacific. Washington: IEEE Computer Society, 2004. 73–82.
11 S.Arbaoui F.Oquendo.Software process modelling and technology.Taunton, UK: Research Studies Press Ltd, 1994: 249–278.
12 Bandinelli S, Fuggetta A, Ghezzi C, et al. Software process modelling and technology. Taunton, UK: Research Studies Press Ltd, 1994: 223–247.
13 Yang Q, Li M, Wang Q, Yang G, et al. An algebraic approach for managing inconsistencies in software processes. Proc. of the Int'l Conf. on Software Processes. Berlin: Springer-Verlag, 2007. 121–133.
14 杨国伟,杨秋松,翟健,等.基于代数的软件过程建模系统的设计与实现.计算机工程与设计,2008,29(3):530–539.
15 Arbaoui S, Derniame J, Oquendo F, et al. A comparative review of process-centered software engineering environments. Annal of Software Engineering, 2002, 14(4): 311–340.
16 Kaiser GE. Experience with marvel. Proc. of the 5th international software process workshop on Experience with software process models. Los Alamitos, CA, USA: IEEE Computer Society Press, 1990: 82–88.
17 Finkelstein A. Software process modelling and technology. New York, NY, USA: John Wiley & Sons, Inc, 1994: 45–68.