软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2011,22(1):1−16 [doi: 10.3724/SP.J.1001.2011.03769]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

一种形式化的组件化软件过程建模方法*

翟 健[1,2+], 杨秋松[1], 肖俊超[1], 李明树[1,3]

[1](中国科学院 软件研究所 互联网软件技术实验室,北京 100190)

[2](中国科学院 研究生院,北京 100049)

[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)

Formalized Approach for Componentized Software Process Modeling

ZHAI Jian[1,2+], YANG Qiu-Song[1], XIAO Jun-Chao[1], LI Ming-Shu[1,3]

[1](Laboratory for Internet Software Technologies, Institute of Software, The Chinese Academy of Sciences, Beijing 100190, China)

[2](Graduate University, The Chinese Academy of Sciences, Beijing 100049, China)

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

+ Corresponding author: E-mail: zhaijian@itechs.iscas.ac.cn

Zhai J, Yang QS, Xiao JC, Li MS. Formalized approach for componentized software process modeling. *Journal of Software*, 2011,22(1):1−16. http://www.jos.org.cn/1000-9825/3769.htm

Abstract: To address the problems of current approaches in software process reuse, in particular the low efficiency in reuse for the operational rules and for the lack of a precise definition of process components, a formalized approach for componentized software process modeling (CSPM) is presented in this paper. CSPM provides a mechanism to support the formal definition of reusable software process components and presents a series of rules to turn process components into a process model. By using CSPM, the reusage of process components can be conducted in a rigorous manner, and the potential errors caused by ambiguity in traditional non-formal modeling methods can be effectively avoided. CSPM can also turn the verification of a combined process model, against certain properties, into a series of sub-problems into its own corresponding components, making an original infeasible problem, under certain circumstances, into feasible ones by exponentially reducing the state space needed to be explored.

Key words: software process; process modeling; process reuse; process component; formal method

摘 要: 为了解决当前软件过程重用方法中存在的问题,特别是由于缺乏对软件过程组件及其操作法则的精确定义所带来的重用中的低效率问题,介绍了一种形式化的组件化软件过程建模方法(componentized software process modeling,简称 CSPM).CSPM 提供了形式化定义可重用软件过程的机制,并且给出了将过程组件组合成过程模型的一系列操作法则.利用 CSPM 方法,能够以严格的方式对软件过程组件进行重用,并且有效地避免了传统非形式化建模方法中因歧义而有可能引起的潜在错误.CSPM 还可以将对组装后的软件过程模型针对某些特定性质的验证问题转化成对其对应组件的一系列子验证问题,从而通过指数地减少需要搜索的状态空间规模,将原来在某些特定环

* 基金项目: 国家自然科学基金(90718042, 60903051); 国家高技术研究发展计划(863)(2007AA010303, 2007AA01Z186); 国家重点基础研究发展计划(973)(2007CB310802)

收稿时间: 2009-04-01; 修改时间: 2009-07-21; 定稿时间: 2009-10-22

境下不实用的验证问题简化成验证代价较小的一系列问题.

关键词: 软件过程;过程建模;过程重用;过程组件;形式化方法
中图法分类号: TP311 文献标识码: A

软件过程重用是软件过程建模研究领域的重要问题之一[1−4].该问题来源于软件工程领域的“软件重用”概念.与软件重用机制所带来的优点类似,良好的软件过程重用机制能够使软件组织快速而高效地建立软件过程模型,有效地降低软件过程建模和分析的成本,提高软件产品质量[2].Hollenbach 指出,使用良好的软件过程重用机制,大致可以使软件过程建模时间减少一个数量级[1].过程重用方法一般要求软件组织拥有自己积累的或第三方提供的、已证明成功的可重用软件过程片段.这里的可重用软件过程片段一般以自然语言或其他形式化方法来描述,针对软件过程中经常遇到的一个或多个实际问题,结构化地描述一个经过验证能够成功解决问题的软件开发活动序列.正确地使用这些过程片段,对实际的过程建模活动能够起到一定的指导作用.在工业界,规模相对较大、成熟度相对较高的软件组织往往拥有自己的过程模板或标准过程,这些过程资产大多都是经过软件组织实际项目或其他专门研究机构检验成功的过程片段[1].这些过程片段一般是为解决实际软件过程中经常遇到的典型问题所提供的特定操作步骤和处理方法.在既有的软件过程片段的基础上进行软件过程建模,是过程重用的一种典型表现形式.

虽然软件过程资产传递了大量软件过程可重用信息,能够高效地解决实际中常见的典型问题,但在当前一般的软件组织中,过程资产并没有充分发挥其应有的作用:首先,一个组织内部所积累的过程资产难以全面覆盖软件过程的所有问题,而不同软件组织中积累的过程资产的数量、粒度、描述方法、侧重点各有不同,这限制了不同组织之间的交流,使得不同软件组织面对相同或相似问题时往往要自行摸索和实验,从而影响了行业的总体效率;第二,由于重用过程片段描述方法不统一,同时描述方法的形式化程度不高,人们在使用这些过程片段时需要依靠自己的主观理解,因此容易导致错误、误解、低效率等情况的发生[1,5];第三,实际应用中缺乏能够保证从成功的过程片段出发、建立成功的软件过程模型的方法,这就限制了软件组织过程资产的使用,使过程资产只能成为过程建模的参考,而不能发挥更大的作用,这也是现有软件过程重用方法的弱点之一.

从以上几点出发,本文提出一种形式化的组件化软件过程建模方法 CSPM(componentized software process modeling).CSPM 以经过实践检验的可重用软件过程组件作为基本的建模单元,在此基础上,详细定义对组件的操作法则,从而使这种软件过程重用方法有良好的可操作性,最大程度地避免了人们在使用软件过程重用机制建模时,由于理解、经验等的差异所可能导致的建模错误,使得建模过程能够被严格地重复和回溯.这里的“软件过程组件”是指经过长期实践验证的、相对标准化的、用以解决特定软件过程问题的软件开发活动序列.其次,在这一方法中,组件内部行为使用基于多元π演算[6]的图形化软件过程建模语言 TRISO/ML[7]进行描述,具有坚实的形式化基础,可以针对活动存在性、活动时序关系等过程性质进行有效的分析和验证;并且可以根据软件过程模型组件化建模的特点,有效地降低过程模型性质验证的需搜索的状态空间规模,从而提高模型检测效率,增强该方法的实用程度.

本文第 1 节回顾相关研究的进展情况.第 2 节介绍 TRISO/ML 软件过程建模语言,该语言是本建模方法所使用的软件过程组件和软件过程模型的描述工具.第 3 节具体介绍组件化软件过程建模方法,包括软件过程组件的定义、操作法则和模型验证方法.第 4 节给出这一方法的应用示例.第 5 节做出总结.

1 相关研究

软件过程研究起源于“软件过程也是软件”[8,9]这一命题的提出.经过数十年的发展,软件过程相关的理论和方法已经为学术界和工业界广泛接受.软件过程是指用于开发和维护软件产品的一系列有序活动,每个活动包括相关的制品、资源(人或其他资源)、组织结构和约束的描述[10].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,并且通过对该抽象描述的分析增强对过程的理解,指导实际软件过程的执行.近 20 余年以来,研究者提出了数十种不同的软件过程建模语言,每种建模语言都有其各自的特点与适用场合,但均不

足以覆盖所有的软件过程建模需求.迄今为止,尚无能够系统、高效地支持软件过程重用机制的软件过程建模语言[11].

软件过程重用是软件工程研究领域的重要研究问题之一.同软件重用一样,与传统的开发方式相比,软件过程重用能够减少大型软件组织生产软件产品的成本和时间[2].CMMI 模型作为一个被广泛使用的软件组织评估参考模型,要求具有较高成熟度的软件组织定义并使用自己的可重用软件过程,从而使软件过程改进更好地执行.最初的软件过程重用仅仅是对可重用的软件过程片段的存储和重现,使用户能够在建立新的过程模型时,从组织的过程资产中"拷贝/修改"相应的过程片段[2].自20世纪90年代后期以来,软件过程重用领域受到了越来越广泛的关注,出现了一系列软件过程模型重用方法或支撑工具[1−5,12−17].Hitchings 和 Perry 指出了软件过程重用方法的基本概念以及需要拥有的特性,例如多维度描述、抽象机制、粒度级别、统一描述、正确性验证等[2,12,18].

由 Ambler 提出的"过程模式(process pattern)"方法是最具影响力的软件过程重用方法之一[3−5,13].Ambler 将过程模式定义为"描述用于软件开发的被证明成功的一种方法和/或一系列活动(a pattern which describes a proven, successful approach and/or series of actions for developing software)".他将过程模式分成3类,从微观到宏观分别为任务过程模式(task process pattern)、阶段过程模式(stage process pattern)和总体过程模式(phase process pattern).阶段过程模式可能由一个或多个任务过程模式及其他元素构成;总体过程模式则可以由任务过程模式和阶段过程模式综合构成.所有过程模式都是由自然语言描述的,与其他模式类似,过程模式也被以文档的形式记录.一个过程模式文档含有多种元素,例如名字、内容、类型、初始条件、解决方案、结束条件、相关模式、已知应用或示例等等.另外,Ambler 还介绍了一部分"反过程模式(process antipattern)"作为过程模式的补充.反过程模式指出了软件过程中针对某些问题常见的错误做法,需要实践者在现实环境中尽量加以避免.通过过程模式,软件过程的制定者或项目管理人员能够在设计软件过程时有所参考,提高软件过程的质量.

虽然过程模式能够描述和记录针对特定问题处理的软件过程片段,从而支持软件过程建模,但仍然存在一些缺陷:首先,有研究者指出,过程模式的最大弱点就在于它是基于自然语言描述的,有可能导致软件过程定义中的不确定、歧义、错误、低效率等问题[4,5];第二,过程模式的问题还在于其没有给出模式操作规则,例如,单一模式中的条件、步骤等可否裁剪或添加,裁剪或添加的条件如何,多模式结合的方法和约束等,都没有详细地指明.因此,在对过程模式的使用过程中,对它的操作将完全依赖于模式使用者对过程模式的理解,这就大大降低了过程模式在指导软件开发过程中的实际意义;第三,过程模式本身是内容相关的,使用过程模式只能对过程中特定的问题提供支持,而对新问题或交叉的问题处理起来弹性显得不够,这也降低了过程模式的实用性.

另一种受关注的过程重用方法称为"工作流模式(workflow pattern)"[14−17].工作流模式是指在工作流应用中,针对经常发生的问题而被证明成功的解决方案.与过程模式相比,该方法只关注工作流要素的结合方式,而不关注其语义,重用的对象是过程的结构,而不是过程本身.Aalst 和 Russell 等人系统地归类、整理了常见的工作流模式,他们将工作流模式分成控制流模式(control-flow pattern)[14]、数据模式(data pattern)[15]、资源模式(resource pattern)[16]和异常处理模式(exception handling pattern)[17].这些工作流模式不能被用来解决软件过程中的具体问题,需要进一步实例化后才能使用,可以看作是对可重用软件过程的进一步抽象.由于其抽象程度过高,对软件过程建模难以起到直接的指导作用.

2 TRISO/ML 软件过程建模语言

本方法中的软件过程组件及相应的软件过程模型都采用 TRISO/ML 软件过程建模语言进行描述.提出 TRISO/ML 语言的初衷是支持集成化的软件过程框架 TRISO Model(tridimensional integrated software development model)[7,19].选择该语言作为软件过程组件的描述工具主要基于以下两点:第一,该语言是一种图形化的过程建模语言,模型中的每个对象都可以具有一定的行为,且可以使其通过特定方式与其他对象进行交互和同步,符合软件过程的特点,形式上比较直观,方便使用;第二,该语言具有严格的基于多元π演算的操作语义,因此可以很好地支持软件过程的执行、分析(包括基于模型检测的软件过程正确性检查和过程间的相互匹配)、演化和分布式执行.一个用TRISO/ML语言描述的软件过程模型可以根据一系列的转换规则,自动地生成多元π

演算的表达式组,用以描述图形化过程的操作语义;继而在多元π演算的基础上,可以支持软件过程的执行和一系列的分析工作.

TRISO/ML 语言采用面向行为的方式来描述软件过程,过程活动间的关系可以包括顺序、并发和选择.活动间还可以通过通道进行信息传递和同步.另外,还可以通过注释的方式表达属性等其他信息.关于 TRISO/ML 语言的定义及其转换成多元π演算表达式所需的转换规则,可以参见文献[20].

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行为之间的顺序、并发和选择关系;用椭圆矩形表示过程活动.本文下面将表示过程活动行为之间关系的图形化符号称为关系运算符,将表示过程活动的图形化符号称为活动节点或简称为活动;关系运算符和活动节点之间用连线连接,表示其上下层关系,称为连接;关系运算符和活动节点统称为节点;关系运算符、活动节点、连接等统称为过程元素.

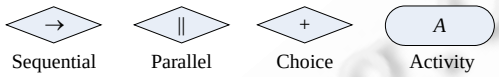


Fig.1 Graphical notations of TRISO/ML

图 1 TRISO/ML 语言图形化符号

3 组件化软件过程建模

组件化软件过程建模方法的核心是软件过程组件的定义及基于软件过程组件的建模操作法则的定义.基于这两方面的定义,人们能够严格地描述并记录从一组软件过程组件到软件过程模型的建模过程.同时,该过程是可重复、可回溯的,为软件过程建模过程的重用及软件过程模型的错误验证及追溯提供了可能,为软件过程的改进提供了新的途径.

3.1 软件过程组件的形式化表示

软件过程组件可以看作具有特定功能的软件过程片段,该片段必须具有结构化的接口,用来与其他软件过程组件交互.同时,根据需要,还可以具有输入、输出通道,用来传递过程文档或其他信息.一个软件组织中的多个软件过程组件及一个软件过程组件中的多个过程活动均可以根据名称加以区分.图形化地,软件过程组件可以表示成如图 2(a)、图 2(b)的形式,其中包括软件过程组件的边界、边界外的接口及输入/输出、边界内的软件过程组件名称及其内部行为.图中的矩形方框表示软件过程组件的边界,其内部字符表示该软件过程组件的名称(如图 2(a)中的字符 *SPC*、图 2(b)中的字符 *WTF* 等);在软件过程组件边界以外,倒三角形表示软件过程组件的接口,端点为空心圆圈的连接表示所有从外部环境输入到软件过程组件的信息的集合,端点为实心圆圈的连接表示所有从软件过程组件输出到外部环境的信息的集合.根据建模需要,软件过程组件内部行为可以显示或隐藏.当需要显式地表达软件过程组件内部行为时,可以使用 TRISO/ML 图表示软件过程活动,以过程开始活动 *s* 和结束活动 *e* 作为接口,与其他软件过程组件连接.

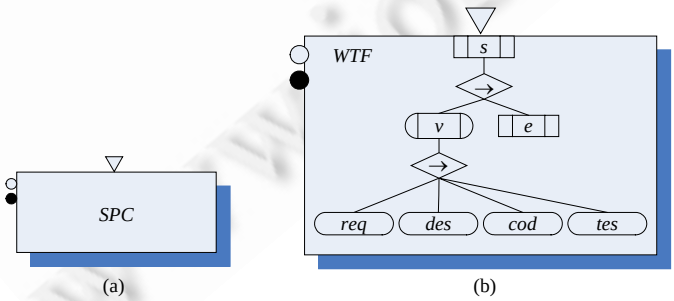


Fig.2 Graphical software process component

图 2 图形化软件过程组件

定义 1(软件过程组件,SPC). 软件过程组件为一多元组(P,s,e,i,o),其中:P 为软件过程组件名称,同时代表该软件过程的内部活动;s 和 e 表示该软件过程组件与外界的接口;i 和 o 分别表示该软件过程组件的输入和输出通道.一个软件过程组件可以简称为 P.进一步地,如果关注软件过程组件的内部行为,该组件内部行为 P 可以分解成多元组 $P=(\Gamma,\delta,\varepsilon,\mu,\Omega)$,其中:

- $\Gamma=\Gamma_a\cup\Gamma_r$ 为组件内部节点集合:Γ_a 为活动节点的集合,$\Gamma_r=\{r|r\in\{\gamma^{\rightarrow},\gamma^{\parallel},\gamma^{+}\}\}$,为关系运算符集合($\gamma^{\rightarrow},\gamma^{\parallel},\gamma^{+}$分别表示顺序关系、并行关系和选择关系).不妨设集合中元素以元素名称区分.
- $\delta\in\Gamma_a\times\Gamma_a$ 为活动节点偏序关系集合:若$\langle a,b\rangle\in\delta,a,b\in\Gamma_a,a\neq b$,则称活动 a 与活动 b 相邻,且活动 a 位于活动 b 之前,记为 $a\succ b$.显然,$(\Gamma_a,\succ)$为偏序关系,该关系在集合Γ_a 上可传递.在一个表示相邻节点的二元组中,二元组左边元素所代表的节点称为右边元素所代表的节点的前驱节点;二元组右边元素所代表的节点称作二元组左边元素所代表的节点的后继节点.
- $\varepsilon\in\Gamma_a\times\Gamma_r\cup\Gamma_r\times\Gamma_a$ 为组件连接集合.
- $\mu\in\Gamma_a\cup\Gamma_r\times\Gamma_a\rightarrow Attr$ 为组件元素性质集合.
- $\Omega=\Gamma_a\rightarrow\{f,v,s,e\}$为活动节点性质映射集合:$f$ 表示功能性节点,v 表示虚节点,s 表示开始节点,e 表示结束节点.该映射为单射,即:针对过程组件Γ_a 中的某一特定活动节点 a,若$\Omega_p(a)$表示其活动节点性质,我们有$\forall a\in\Gamma_a,\Omega_p(a)\in\{f,v,s,e\}$;相应地,集合中某一特定类型活动节点的集合可以表示为

$$\Omega_p^{-1}(m)=\{a\mid\Omega_p(a)=m,m\in\{f,v,s,e\}\}.$$

为简化研究问题,不妨设每个软件过程组件有且只有唯一的开始节点和结束节点.

软件过程组件的内部行为可以用扩展的 TRISO/ML 图形方式表示.根据上述定义,TRISO/ML 语言中原有的过程活动可以被进一步细化成 3 类活动:功能性活动、虚活动、开始/结束活动,并且通过使用不同的图形符号加以区分,如图 3 所示.

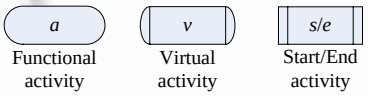


Fig.3 Graphical notations of extended TRISO/ML

图 3 扩展的 TRISO/ML 语言图形化符号

例如:如图 2(b)所示的软件过程模型(WTF,s,e,i,o)表示软件开发活动的瀑布模型,该组件可以通过输入通道 i 和输出通道 o 与外界交互信息,并且可以通过接口 s 触发其内部行为,顺序地执行需求(req)、设计(des)、编码(cod)和测试(tes)这 4 个活动,执行结束后,通过接口 e 向外部环境发出信号.

其内部行为可以表示为 $WTF=(\Gamma,\delta,\varepsilon,\mu,\Omega)$,其中:

- $\Gamma=\{s,v,e;req,des,cod,tes;r_1^{\rightarrow},r_2^{\rightarrow}\}$;
- $\delta=\{\langle s,v\rangle,\langle v,req\rangle,\langle req,des\rangle,\langle des,cod\rangle,\langle cod,tes\rangle,\langle tes,e\rangle\}$;
- $\varepsilon=\{s\times r_1^{\rightarrow},r_1^{\rightarrow}\times v,r_1^{\rightarrow}\times e,v\times r_2^{\rightarrow},r_2^{\rightarrow}\times req,r_2^{\rightarrow}\times des,r_2^{\rightarrow}\times cod,r_2^{\rightarrow}\times tes\}$;
- $\mu=\varnothing$;
- $\Omega=\{s\rightarrow s,e\rightarrow e,v\rightarrow v,\{req,des,cod,tes\}\rightarrow f\}$.

基于δ上的传递性,集合δ可以进一步简化,表示为$\delta=s\succ v\succ req\succ des\succ cod\succ tes\succ e$.

为简化叙述,下面在软件过程组件定义的基础上,针对其内部行为定义若干辅助函数.

定义 2(连接的端点元素集合). 一个软件过程组件中,连接发出的活动节点或关系运算符集合为

$$\overline{e}=\{a\mid\exists e=a\times r,a\in\Gamma_a,r\in\Gamma_r,e\in\varepsilon\}\cup\{r\mid\exists e=r\times a,a\in\Gamma_a,r\in\Gamma_r,e\in\varepsilon\}.$$

连接到达的过程元素集合为

$$\underline{e}=\{a|\exists e=r\times a,a\in\Gamma_a,r\in\Gamma_r,e\in\varepsilon\}\cup\{r|\exists e=a\times r,a\in\Gamma_a,r\in\Gamma_r,e\in\varepsilon\}.$$

定义 3(以某元素为端点的连接集合). 一个软件过程组件中,以某一过程元素 m 为起点的连接集合为

$$\overline{m}=\{e\mid\exists e=m\times r,m\in\Gamma_a,r\in\Gamma_r,e\in\varepsilon\}\cup\{e\mid\exists e=m\times a,m\in\Gamma_r,a\in\Gamma_a,e\in\varepsilon\}.$$

相应地,以某一过程元素 n 为终点的连接集合为

$$\underline{n}=\{e|\exists e=r\times n,n\in\Gamma_a,r\in\Gamma_r,e\in\varepsilon\}\cup\{r|\exists e=a\times n,n\in\Gamma_r,a\in\Gamma_a,e\in\varepsilon\}.$$

定义 4(改变节点活动性质). 表达式 $a\to m\triangleright n$ 表示将软件过程组件中 P 中的活动节点 a 的活动属性由 m 更新为 n,即 $a\to m\triangleright n=\Omega_p-\{a\to m\}+\{a\to n\}$,其中,$a\in\Gamma_a,m,n\in\{f,v,s,e\}$.进一步地,为了表述简便,引入集合操作表达式:

$$\{a_1,\ldots,a_i\}\to m\triangleright n=(a_1\to m\triangleright n)\wedge\ldots\wedge(a_i\to m\triangleright n).$$

3.2 图形化软件过程组件建模操作定义

根据实际建模需要,对软件过程组件施加的操作至少应包括如下 5 种:连接;添加、删除;扩展、抽象.其中:除连接操作外,其余操作均涉及软件过程组件的内部行为;添加与删除、扩展与抽象分别互为逆操作.上述 5 种操作通过复合,能够覆盖一般的软件过程建模过程.对于某些特殊操作,当用上述 5 种操作描述较为复杂时,也可以定义新的操作法则,添加到建模中,不会对建模方法总体结构产生整体影响.形式化地定义对软件过程组件的操作,首先可以保证该操作本身的可理解性,避免了人们在使用软件过程组件建模时,由于主观原因造成的对操作法则的误解,进而降低了在软件过程建模中产生错误的可能性;其次,保证了使用该操作法则对软件过程组件进行操作后,软件过程组件的结构和形式化特性能够正确地保持.由于这些操作法则既可以用来建立软件过程模型,又可以用来维护软件过程组件,这实际上统一了软件过程组件和软件过程模型的结构,有助于灵活而富有弹性地对软件过程片段进行重用:可重用的软件过程片段都可以作为软件过程组件,用来组合构造更大的软件过程模型;软件过程组件也可以作为一个功能相对单一的软件过程模型单独使用.

定义 5(连接,$P_1\oplus^r P_2$). 两个软件过程组件$(P_1,s_1,e_1,i_1,o_1),(P_2,s_2,e_2,i_2,o_2)$连接,需要加入新节点 $s,e,v,r^{\to}$,并且根据两组件的执行时序关系,添加时序关系符号 $\tilde{r}(\tilde{r}\in\{r^{\to},r^{\parallel},r^{+}\})$ 以及相应的连接,得到的软件过程组件如图 4(a)所示.合并两个软件过程组件的输入/输出通道,可以用$(P',s,e,i_1\cup i_2,o_1\cup o_2)$表示连接后的软件过程组件.对于软件过程组件 P',可以继续使用软件过程组件的操作进行处理.进一步地,关注软件过程组件的内部行为,设两个软件过程组件的内部行为分别为 $P_1=(\Gamma_1,\delta_1,\varepsilon_1,\mu_1,\Omega_1)$和 $P_2=(\Gamma_2,\delta_2,\varepsilon_2,\mu_2,\Omega_2)$,则连接后的软件过程组件的内部行为可以表示为 $P'=(\Gamma',\delta',\varepsilon',\mu',\Omega')$,其中:

- 节点集合Γ'要求在包含两组件节点集合中全部元素的基础上增加开始节点、结束节点、虚节点、顺序关系运算符、组件连接关系运算符这 5 个过程元素,即 $\Gamma'=\Gamma_1\cup\Gamma_2\cup\{s,e,v,r^{\to},\tilde{r}\}$;
- 顺序关系集合δ'要求在包含两组件活动节点全部顺序关系的基础上增加新加入活动节点之间的顺序关系以及两组件各活动节点间的顺序关系,即$\delta'=\delta_1\cup\delta_2\cup\{\langle s,v\rangle,\langle v,s_1\rangle,\langle e_1,s_2\rangle,\langle e_2,e\rangle\}$;
- 连接集合ε'要求在包含两组件全部连接元素的基础上增加新加入节点之间的连接以及指向两组件开始节点的连接,即 $\varepsilon'=\varepsilon_1\cup\varepsilon_2\cup\{s\times r^{\to},r^{\to}\times e,r^{\to}\times v,v\times\tilde{r},\tilde{r}\times s_1,\tilde{r}\times s_2\}$;
- 属性集合μ'要求包含两组件所有属性元素,即$\mu'=\mu_1\cup\mu_2$;
- 活动节点性质集合Ω'要求在包含两组件所有活动节点性质的基础上,将两组件中的原开始/结束活动的节点属性变更为虚节点,同时赋予新添加的节点以各自的属性,即

$$\Omega'=\Omega_1\cup\Omega_2+\{\{s_1,s_2\}\to s\triangleright v\}+\{\{e_1,e_2\}\to e\triangleright v\}+\{v\to v\}+\{s\to s\}+\{e\to e\}.$$

软件过程组件连接操作可以表示为 $P_1\oplus^r P_2$,读作“将软件过程组件 P_1 和 P_2 以 r 关系相连接”,其中,$r\in\{\gamma^{\to},\gamma^{\parallel},\gamma^{+}\}$,具体连接运算符用脚标表示.该操作可以简化表示为$(\oplus^r,P_1/P_2)$.

例如,若图 4(b)和图 4(c)表示两个软件过程组件,分别将其命名为 P_1 和 P_2,则执行操作 $P_1\oplus^{\to}P_2$ 将得到的软件过程组件,如图 4(d)所示.

定义 6(添加, $\boldsymbol{P+\overline{b/a}/c}$). $P=(\Gamma,\delta,\varepsilon,\mu,\Omega)$为一个软件过程组件的内部行为,向该组件中相邻的活动节点 b 和 c 之间添加一个新的功能性活动 $a(b,c\in\Gamma_a$ 且$\langle b,c\rangle\in\delta)$,使之与活动节点 b 并列,可以得到新的软件过程组件,其内部行为可以表示为 $P'=(\Gamma',\delta',\varepsilon',\mu',\Omega')$,其中:

- 在集合Γ_a中添加新元素 a,其他保持不变,即$\Gamma'=(\Gamma_a+a)\cup\Gamma_r$.

- 在集合δ中添加二元组$\langle b,a\rangle$和$\langle a,c\rangle$,并删除$\langle b,c\rangle$,即$\delta'=\delta+\langle b,a\rangle+\langle a,c\rangle-\langle b,c\rangle$.
- 在集合ε中添加一条连接,该连接的起点为指向活动节点 b 的连接的起点,终点为活动节点 a,即

$$\varepsilon'=\varepsilon+r\times a(\{r\mid \underline{b}\subseteq \bar{r}\}).$$

- 在集合μ中添加新活动节点 a 及新连接 $\underline{a}$ 所带的属性元素,即$\mu'=\mu+Attr(a)+Attr(\underline{a})$.
- 在集合Ω中添加新活动节点 a 的属性信息,即$\Omega'=\Omega+\{a\to f\}$.

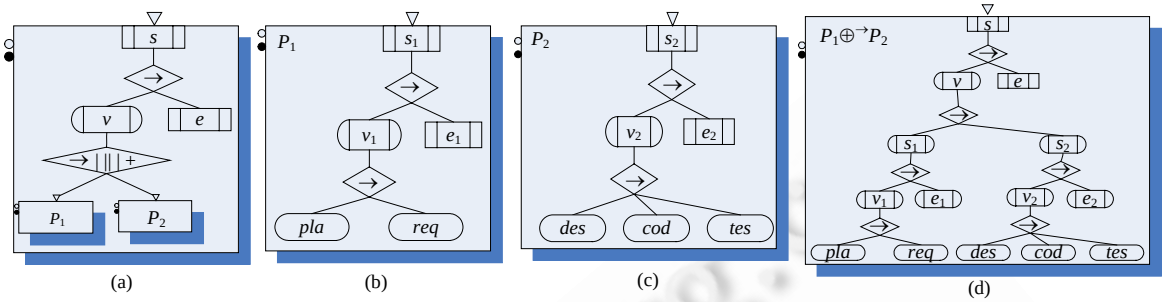


Fig.4 Connection operation of software process components

图 4 软件过程组件的连接操作

添加操作不影响软件过程组件的接口和输入/输出通道,在不关注软件过程组件内部行为的情况下,外界不能观察到该组件的变化.添加操作可以表示为 $P+\overline{b/a}/c$,读作"在过程组件 P 中,b,c 两活动之间添加与活动 b 并列的活动 a".在操作对象明确的情况下,本操作可以简化表示为 $(+,\overline{b/a}/c)$.同样地,表达式 $P+b/\overline{a/c}$ (在过程组件 P 中,b,c 两个活动之间添加与活动 c 并列的活动 a)或 $(+,b/\overline{a/c})$ 可以类似定义.若活动 b 和 c 已构成并列的活动,则"在过程组件 P 中,b,c 两个活动之间添加活动 a"可以表示为 $P+\overline{b/a/c}$;在不产生歧义的情况下,可以简化表示为 $P+b/a/c$.

例如,执行操作 $WTF_1=WTF+v/\overline{pla/req}$,得到如图 5 所示的软件过程组件,其内部行为可表示为 $WTF_1=(\Gamma,\delta,\varepsilon,\mu,\Omega)$,其中:

- $\Gamma=\{s,v,e;pla,req,des,cod,tes;r_1^{\to},r_2^{\to}\}$;
- $\delta=s\succ v\succ pla\succ req\succ des\succ cod\succ tes\succ e$;
- $\varepsilon=\{s\times r_1^{\to},r_1^{\to}\times v,r_1^{\to}\times e,v\times r_2^{\to},r_2^{\to}\times pla,r_2^{\to}\times req,r_2^{\to}\times des,r_2^{\to}\times cod,r_2^{\to}\times tes\}$;
- $\mu=\varnothing$;
- $\Omega=\{s\to s,e\to e,v\to v,\{pla,req,des,cod,tes\}\to f\}$.

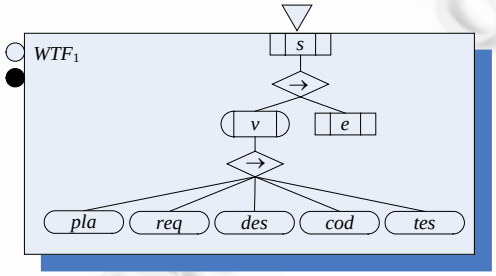


Fig.5 WTF_1: Plan activity added software process component WTF

图 5 添加计划活动后的瀑布模型过程组件

虽然软件过程组件 WTF_1 的内部行为在形式上与图 4(d)不同,但其语义完全一致.之所以会在形式上产生区别,是因为连接两个软件过程组件的操作加入了旨在保持模型结构化特征的虚节点.

定义 7(删除,$P-a$). $P=(\Gamma,\delta,\varepsilon,\mu,\Omega)$为一种软件过程组件的内部行为,删除其中的功能性活动 $a(a\in\Gamma_a)$,可以得到新的软件过程组件 P'.

若 P 中与功能性活动 a 并列的其他功能性活动大于等于两个,则 P'的内部行为可以表示为 $P'=(\Gamma',\delta',\varepsilon',\mu',\Omega')$,其中:

- 从集合Γ中删除活动节点 a,其他元素保持不变,即$\Gamma'=(\Gamma_a-a)\cup\Gamma_r$.
- 从集合δ中删除与活动 a 有关的二元组,并添加新的顺序关系元素,其他元素保持不变.若节点 a 前驱和后继的节点分别为 b 和 c,则需要删除元素$\langle b,a\rangle,\langle a,c\rangle$,添加元素$\langle b,c\rangle$,即
$$\delta'=\delta-\langle b,a\rangle-\langle a,c\rangle+\langle b,c\rangle(\langle b,a\rangle,\langle a,c\rangle\in\delta).$$
- 从集合ε中删除以活动 a 为终点的连接元素,其他元素保持不变,即$\varepsilon'=\varepsilon-\underline{a}$.
- 从集合μ中删除活动 a 和连接 $\underline{a}$ 的所带的属性元素,其他元素保持不变,即$\mu'=\mu-Attr(a)-Attr(\underline{a})$.
- 从集合Ω中删除活动 a 的属性,其他元素保持不变,即$\Omega'=\Omega-\{a\to f\}$.

若 P 中与功能性活动 a 并列的功能性活动只有一个,并且名为 a',则 P'的内部行为可以表示为 $P'=(\Gamma',\delta',\varepsilon',\mu',\Omega')$,其中:

- 从集合Γ中删除活动元素 a,控制活动 a 的时序关系运算符 r 以及控制时序关系运算符 r 的虚活动 v,其他元素保持不变,即$\Gamma'=(\Gamma_a-a-v)\cup(\Gamma_r-r)$,其中,$\underline{a}\subseteq\overline{r}$,$\underline{r}=\overline{v}$.
- 从集合δ中删除与活动a,v有关的二元组,并添加新的顺序关系元素,其他元素保持不变.若节点a前驱和后继的节点分别为 b 和 c,节点 v 前驱和后继的节点分别为 d 和 e,则需要删除元素$\langle b,a\rangle,\langle a,c\rangle,\langle d,v\rangle,\langle v,e\rangle$,添加元素$\langle b,c\rangle,\langle d,e\rangle$,即$\delta'=\delta-\langle b,a\rangle-\langle a,c\rangle-\langle d,v\rangle-\langle v,e\rangle+\langle b,c\rangle+\langle d,e\rangle(\langle b,a\rangle,\langle a,c\rangle,\langle d,v\rangle,\langle v,e\rangle\in\delta)$.
- 从集合ε中删除以活动 a、v、关系运算符 r 为起点或终点的连接元素,添加从控制 v 的关系运算符 r' 到 a'的连接,其他元素保持不变,即 $\varepsilon'=\varepsilon-\overline{r}-\underline{r}-\underline{v}+\varepsilon_s,\varepsilon_s=\{e\mid e=r'\times a',\underline{v}\subseteq\overline{r'}\}$.
- 从集合μ中删除活动 a,v 和连接 $\underline{a}$,$\underline{v}$,$\overline{v}$ 所带的属性元素、添加连接 $\underline{a'}$的属性元素,其他元素保持不变,即 $\mu'=\mu-Attr(a,v)-Attr(\underline{a},\underline{v},\overline{v})+Attr(\underline{a'})$.
- 从集合Ω中删除活动 a 和 v 的属性,其他元素保持不变,即$\Omega'=\Omega-\{a\to f\}-\{v\to v\}$.

删除操作不影响软件过程组件的接口和输入/输出通道,在不关注软件过程组件内部行为的情况下,外界不能观察到该组件的变化.删除操作可以表示为 $P-a$,读作"从软件过程组件 P 中删除活动 a".在操作对象明确的情况下,本操作可以简化表示为$(-,a)$.

删除操作是添加操作的逆操作.对如图 5 所示的软件过程组件的内部行为执行操作 WTF_1-pla,即可将过程组件 WTF_1 的内部行为还原成形如图 2(b)所示的软件过程组件 WTF 的内部行为.

定义 8(扩展,$P_1\otimes P_2/a$). $(P_1,s_1,e_1,i_1,o_1),(P_2,s_2,e_2,i_2,o_2)$为两个软件过程组件,$P_2$ 的内部行为可以表示为 $P_2=(\Gamma_2,\delta_2,\varepsilon_2,\mu_2,\Omega_2)$,$a$ 为软件过程组件 P_2 的内部活动,即 $a\in\Gamma_2$.用软件过程组件 P_1 扩展软件过程组件 P_2 中的活动 a,可以得到新的软件过程组件$(P',s_2,e_2,i_1\cup i_2,o_1\cup o_2)$.进一步关注软件过程组件 P_1 的内部行为 $P_1=(\Gamma_1,\delta_1,\varepsilon_1,\mu_1,\Omega_1)$,则扩展后的软件过程组件的内部行为为 $P'=(\Gamma',\delta',\varepsilon',\mu',\Omega')$,其中:

- 节点集合Γ'为两组件全部活动节点元素,除去被扩展的活动节点 a,即$\Gamma'=\Gamma_1\cup\Gamma_2-a$.
- 活动节点顺序集合δ'要求包括两个过程组件的全部偏序关系,删除与活动 a 有关的元素,并添加新元素,其他顺序关系保持不变.若活动 a 节点的前驱和后继节点分别为 b 和 c,则需要删除元素$\langle b,a\rangle,\langle a,c\rangle$,添加元素$\langle b,s_1\rangle,\langle e_1,c\rangle$,即$\delta'=\delta-\langle b,a\rangle-\langle a,c\rangle+\langle b,s_1\rangle+\langle e_1,c\rangle(\langle b,a\rangle,\langle a,c\rangle\in\delta)$.
- 连接集合ε'要求包含两组件内的全部连接,删除以活动 a 为终点的连接,添加从指向活动 a 的关系运算符到扩展组件开始节点的连接,即
$$\varepsilon'=\varepsilon_1\cup\varepsilon_2-\underline{a}+\varepsilon_{s_1},\varepsilon_{s_1}=\{e\mid e=r\times s_1,\underline{a}\subseteq\overline{r},r\in\Gamma_1\}.$$
- 元素属性集合μ'要求包含两组件中的所有元素属性,删除与活动节点 a 以及以 a 为终点的连接的属性元素,添加以虚节点 s_1 为终点的连接的属性元素,即$\mu'=\mu_1\cup\mu_2-Attr(a)-Attr(\underline{a})+Attr(\underline{s_1})$.
- 活动性质集合Ω'要求包含两组件的所有节点性质元素,删除活动 a 的节点性质,将扩展组件 P_1 的开始

结束节点的性质变成虚节点,即

$$\Omega'=\Omega_1\cup\Omega_2-\{a\to f\}+\{s_1\to s\triangleright v\}+\{e_1\to e\triangleright v\}.$$

扩展操作可以表示为 $P_1\otimes P_2/a$,读作"用软件过程组件 P_1 扩展软件过程组件 P_2 中的活动节点 a".在操作对象明确的情况下,该操作符号可以简化为$(\otimes,P_1/a)$.

例如,执行操作 $WTF_2=PLA\otimes WTF_1/pla$,可以得到新的软件过程组件 WTF_2,如图 6 所示.

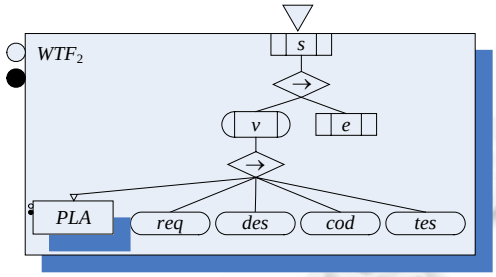
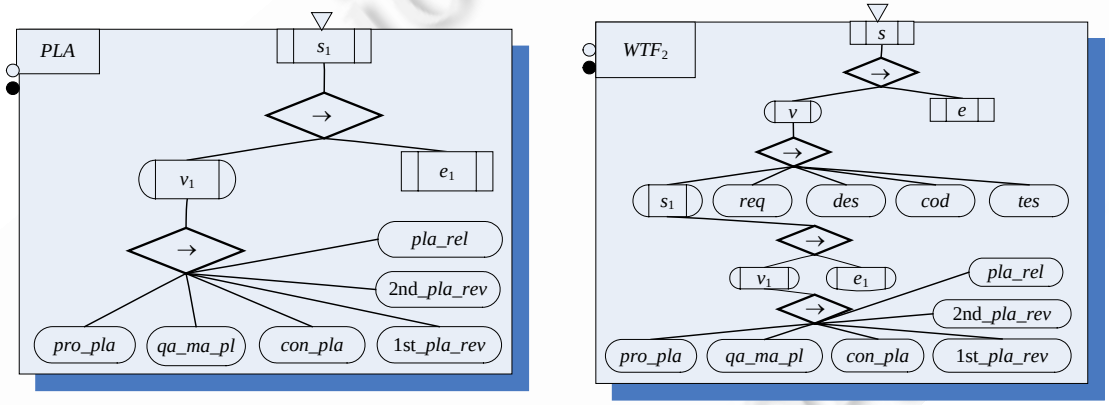


Fig.6 WTF_2: Extended software process component WTF_1

图 6 执行软件过程组件扩展操作的结果

进一步关注软件过程组件 PLA 的内部行为,若过程组件 PLA 的内部行为如图 7(a)所示,则扩展后的过程组件 WTF_2 的内部行为如图 7(b)所示.



(a) (b)

Fig.7 Details of component PLA and WTF_2

图 7 执行计划活动的软件过程组件和扩展后的瀑布模型组件

定义 9(抽象,$P\odot P_1/a$). (P,s,e,i,o)为一软件过程组件,$P=(\Gamma,\delta,\varepsilon,\mu,\Omega)$为其内部行为.$(P_1,s_1,e_1,i_1,o_1)$为 P 中嵌套的软件过程组件,且未被扩展,其内部行为为 $P_1=(\Gamma_1,\delta_1,\varepsilon_1,\mu_1,\Omega_1)$;用功能性活动 a 抽象 P_1 可以得到操作后的软件过程组件$(P',s,e,i-i_1,o-o_1)$,其内部行为可以表示为 $P'=(\Gamma',\delta',\varepsilon',\mu',\Omega')$,其中:

- 节点集合Γ'要求从原节点集合中删除 P_1 中的所有节点,添加抽象后的活动节点 a,即$\Gamma'=\Gamma-\Gamma_1+a$.
- 功能性节点顺序关系集合δ'要求从原顺序关系集合中删除所有与 P_1 中元素有关的二元组,添加$\langle b,c\rangle(\langle b,s_1\rangle+\langle e_1,c\rangle\in\delta_1)$,即$\delta'=\delta-\delta_1+\langle b,c\rangle$.
- 连接集合ε'要求从原连接集合中删除所有与 P_1 有关的连接,添加从连接 P_1 中开始节点的节点 l 到抽象后的功能性节点 a 的连接,即 $\varepsilon'=\varepsilon-\varepsilon_1+l\times a,\underline{s_1}\subseteq\overline{l}$.

- 元素属性集合μ'要求从原元素属性集合中删除与 P_1 有关的属性元素,添加抽象后节点 a 及以其为终点的连接的属性元素,即$\mu'=\mu-\mu_1+Attr(a,l\times a)$.
- 活动节点性质集合Ω'要求从原活动节点性质集合中删除与 P_1 相关的活动节点性质,添加活动 a 的活动性质,即$\Omega'=\Omega-\Omega_1+\{a\to f\}$.

该操作可以表示为 $P\odot P_1/a$,读作"用功能性活动 a 抽象过程组件 P 中过程组件 P_1".在操作对象明确的情况下,该操作可以简化表示为$(\odot,a/P_1)$.

抽象操作是扩展操作的逆操作.例如,软件过程组件 WTF_2 的内部行为可以通过抽象操作还原成软件过程组件 WTF_1 的内部行为.

由上述 5 种软件过程组件的操作法则,软件过程模型的组件化建模过程可以用操作表达式表示;利用相同的软件过程组件,软件过程建模过程可以严格地重复、回溯.例如,从形如图 2(b)所示的软件过程组件 WTF 到形如图 6 所示的软件过程组件 WTF_2 的建模过程可以用如下中缀运算符操作表达式(1)表示:

$$WTF_2 = PLA \otimes (WTF + v / \overline{pla / req}) / pla \tag{1}$$

相应地,该操作的逆操作表达式如表达式(2)所示:

$$WTF=(WTF_2\odot PLA/pla)-pla \tag{2}$$

进一步地,上述两个表达式也可以用简化的操作系列方法加以表示:

中缀运算符表达式(1)可以变形为前缀运算符操作序列(3)的形式,表示该模型来源于软件过程组件 WTF,并且在建模过程中经过了添加、扩展等两个标准操作.具体来说,就是在软件过程组件 WTF 内部行为中活动节点 v 和 req 之间添加了活动节点 pla,然后进一步使用软件过程组件 PLA 扩展活动节点 pla.为了统一表示形式,作为建模起点的软件过程组件可以表示成(ϕ,C)的形式,其中,C 为软件过程组件的名称,ϕ表示无任何操作.

$$WTF_2 = (\phi,WTF),(+,v / \overline{pla / req}),(\otimes,PLA / pla) \tag{3}$$

类似地,表达式(2)表示的建模过程的逆过程也可以简化地表示为操作序列(4)的形式.该操作序列表示软件过程组件 WTF_2 的内部行为顺序地经过一次抽象、一次删除操作能够还原到软件过程组件 WTF 的内部行为所表示的原始瀑布模型.具体来说,就是将 WTF_2 中的软件过程建模组件 PLA 抽象为功能性活动 pla,进而删除上一步得到的功能性节点 pla.

$$(\phi,WTF_2),(\odot,pla/PLA),(-,pla) \tag{4}$$

由于这一方法使用标准的软件过程组件和标准的组件化建模操作,因此可以最大限度地避免软件过程建模中人们的主观因素,避免歧义和误解.进一步地,由于得到标准软件过程组件和操作法则的支持,利用本方法建立软件过程模型的过程能够被严格、准确地记录及客观地重复,有利于自动化建模工具和分析方法的部署.同时,该软件过程模型的建模过程还能够被准确地回溯,有利于对软件过程建模中错误的追查和软件过程组件本身的维护和改进.

3.3 软件过程模型性质验证

3.3.1 软件过程性质描述

由于使用本方法建立的软件过程模型具有严格的多元π演算基础,利用现有的有限状态模型检测工具(如 CWB-NC[21],SPIN[22],MWB[23]等),我们可以从不同角度验证和判断使用本方法所建立的软件过程模型的性质是否成立,进而判断软件过程模型本身的正确性.一般地,我们使用模态μ演算[24]描述一个进程代数系统的系统性质.

基于软件过程模型的多元π演算表达形式,多种模型检测工具可以检测过程模型中是否存在死锁,从而优化过程模型.具体来说,不存在死锁这一过程性质可以用模态μ演算表达式(5)来表示:

$$\nu Z.\langle -\rangle tt\wedge[-]Z \tag{5}$$

如果一个进程代数系统能够满足该性质,则说明该系统不存在死锁.

进一步地,模型中特定活动的存在性及其时序关系作为一类系统性质也可以用模态μ演算表达式表示,并

且使用模型检测工具进行验证.一般地,活动事件 a 存在于一个进程代数系统中,可以用μ演算表达式(6)表示.相对地,活动事件 a 不存在于进程代数系统中,可以用表达式(7)表示:

$$\mu X.[a]tt\wedge\langle-\rangle tt\wedge[-a]X \tag{6}$$

$$\nu X.[a]ff\wedge\langle-\rangle tt\wedge[-a]X \tag{7}$$

通过对上述两类活动事件存在性质表达式的组合,可以进一步表达系统中活动事件之间的时序关系.例如,“如果活动事件 a 存在于系统中,那么活动事件 b 也存在于系统中,并且出现在活动事件 a 之后”,可以表示为表达式(8)的形式:

$$\mu X.[a](\mu Y.[b]tt\wedge\langle-\rangle tt\wedge[-b]Y)\wedge\langle-\rangle tt\wedge[-a]X \tag{8}$$

“如果活动事件 a 存在于系统中,则活动事件 b 必然存在于系统中,并且出现在活动事件 a 之前”,可以表示为表达式(9)的形式:

$$\mu X.[b](\mu Y.[a]tt\wedge\langle-\rangle tt\wedge[-a]Y)\wedge\langle-\rangle tt\wedge[a]ff\wedge[-(a,b)]X \tag{9}$$

通过对上述表达式中 a,b 等变量名的实例化,可以使上述表达式表示模型系统中所要验证的性质,并且通过模型检测工具进行验证,如果该表达式为真,则说明该性质在模型系统中成立,否则不成立.

例如,在上节所示的软件过程模型中,可以用模态μ演算表达式描述过程性质“过程模型中存在计划评审活动”,如果计划评审活动表示为 pla_rev,则该性质可以用表达式(10)表示:

$$\mu X.[pla\_rev]tt\wedge\langle-\rangle tt\wedge[-pla\_rev]X \tag{10}$$

过程性质“需求活动存在于过程中,且发生在计划评审活动之后”可以用表达式(11)表示:

$$\mu X.[pla\_rev](\mu Y.[req]tt\wedge\langle-\rangle tt\wedge[-req]Y)\wedge\langle-\rangle tt\wedge[-pla\_rev]X \tag{11}$$

过程性质“计划评审活动存在于过程中,且发生在计划发布活动之前”,如果计划发布活动表示为 pla_rel,则该性质可以用表达式(12)表示:

$$\mu X.[pla\_rev](\mu Y.[pla\_rel]tt\wedge\langle-\rangle tt\wedge[-pla\_rel]Y)\wedge\langle-\rangle tt\wedge[pla\_rel]ff\wedge[-(pla\_rev,pla\_rel)]X \tag{12}$$

可以看出,描述上述过程性质的模态μ演算表达式都是将上述模态μ演算表达式中的变量名实例化后得到的.当描述其他过程性质时,也可以类似操作.关于模态μ演算的具体细节,可以参见文献[24].

进一步地,基于本方法的组件化特征,一部分对于所建立的软件过程模型性质的验证,可以转化为针对构成该过程模型的软件过程模型组件的验证,从而降低了性质验证的复杂程度,提高了性质验证的速度和准确率.

3.3.2 软件过程性质的组件化验证方法

实际环境中,用模型检测的方法验证软件过程模型的性质所面临的最大问题是模型状态爆炸[25].实际环境中,软件过程模型往往规模比较大、活动数量多、活动之间关系复杂.对于模型检测工具来说,检测大规模系统的时间、空间复杂度会随着并发活动数量的增加以指数形式增长,过大的时间代价使得软件过程模型性质验证在实际工作中难以实用.如果能够降低软件过程模型的复杂度,减少其中的活动数量,则可以减小性质验证时需要搜索的状态空间规模,从而大大提高软件过程模型性质验证的效率,使之更加实用化.

针对组件化软件过程建模的特点,一部分软件过程模型的整体性质可以表示成构成软件过程模型的软件过程模型组件的性质的组合的形式.由于每个软件过程模型组件的活动节点数量都远小于整个软件过程模型的活动节点数量,因而在进行模型检测时,检测软件过程组件性质的复杂度也远小于整体检测软件过程模型性质的复杂度.在取得针对各个软件过程组件的检测结果后,可以将该结果进行组合,从而导出软件过程模型的整体性质.我们在本文中重点针对软件过程模型中的活动存在性和活动时序关系性质进行讨论,之所以研究模型中这两类性质的验证,是因为在实际环境中对软件过程模型性质的验证主要都是围绕着这两类验证进行的,对这两类过程性质的验证能够覆盖实际环境中绝大多数的软件过程模型性质验证的情况.

(1) 软件过程模型的活动存在性

定义 10(活动存在性,$E(\alpha\in M)$). 在一个软件过程模型 M 中,存在名为α的活动.

针对组件化软件过程建模的序列表示形式 $S_M=\{(\bullet,f)\}$,其中,$\bullet\in\{+,-,\oplus,\otimes,\odot,\phi\}$表示组件操作运算符,$f$ 表示二元组中的表达式.我们可以将 $E(\alpha\in M)$按顺序分解成各个过程组件性质组合的形式,该组件性质组合表达式可

称为组件化过程性质验证表达式,简称为性质验证表达式.针对组件化软件过程模型建模的各种操作,该建模序列中可能存在的元素类型和建立性质验证表达式的相应的处理方式见表 1.

Table 1 Compositional construction of properties featuring the existence of activities

表 1 活动存在性质验证表达式的组件化构造

Item in sequence	Corresponding expression	Connection to previous expressions
$(+,a_M^1/act/a_M^2)$	$E(\alpha=act)$	$\vee$
$(-,act)$	$\neg(\alpha=act)$	$\wedge$
$(\oplus,C_1/C_2)$	$E(\alpha\in C_1)\vee E(\alpha\in C_2)$	$\vee$
$(\otimes,C/c)$	$E(\alpha\in C)\wedge\neg E(\alpha=c)$	$\vee$
$(\odot,c/C)$	$\neg E(\alpha\in C)\wedge E(\alpha=c)$	$\vee$
(ϕ,C)	$E(\alpha\in C)$	–

其中, $E(\alpha=act)=\begin{cases}0, & \alpha\neq act\\ 1, & \alpha=act\end{cases}$, $E(\alpha\in C)=\begin{cases}1, & \alpha\in C\\ 0, & \alpha\notin C\end{cases}$.

关系 $a=act$ 可以通过直接比较获得,$\alpha\in C$ 可用模型检测工具验证得到结果.具体而言,从建模操作序列从左向右搜索,若序列中的元素满足上表中的相应形式,则将其对应的性质描述表达式添加到组件化过程性质表达式中;如果需要与之前构造的表达式相连,则使用相应的连接符号.建立的性质验证表达式中,逻辑与($\wedge$)和逻辑或($\vee$)操作符号具有相同的优先级,对表达式的执行按从左到右的顺序进行;仅括号可以改变表达式的执行顺序.不含操作的过程组件(ϕ,C)只可能出现在序列的第 1 项,作为建模的基础组件,因此其无需与之前的表达式相连接,故没有相应的连接方式.若性质验证逻辑表达式的值为真,则表示活动α在软件过程模型 M 中存在;反之,则该活动不存在.若活动α在模型 M 中存在,则可进一步确定活动α在模型 M 中的位置.

定义 11(活动位置,$E_L(\alpha\in M)$). 活动α在模型 M 中的位置为 $E_L(\alpha\in M)$,且

$$E_L(\alpha\in M)=\begin{cases}act, & E(\alpha\in M)=1\wedge E(\alpha=act)=1\\ C, & E(\alpha\in M)=1\wedge E(\alpha\in C)=1\\ \varnothing, & E(\alpha\in M)=0\end{cases}.$$

例如,上述建模操作序列(3): $WTF_2=(\phi,WTF),(+,v/\overline{pla/req}),(\otimes,PLA/pla)$,要检查活动$\alpha$在模型 WTF_2 中的存在性,则可如下表示:

$$E(\alpha\in WTF_2)=E(\alpha\in WTF)\vee E(\alpha=pla)\vee(E(\alpha\in PLA)\wedge\neg E(\alpha=pla)) \quad (13)$$

这样,活动在模型中的存在性问题被简化成分别检查活动在构成模型的组件中是否存在的问题:若等式右边表达式的逻辑值为非空,则说明活动α在过程模型 WTF_2 中存在,反之则不存在.这样的分解大大降低了该性质检测的复杂度,提高了检测效率.

(2) 软件过程模型的活动时序性质

定义 12(活动时序,$T(M|\alpha\rightarrow\beta)$). 在一个软件过程模型 M 中,存在名为α和β的活动,并且名为β的活动位于活动α之后($\alpha\neq\beta$).

针对组件化软件过程建模的序列表示形式,我们可以将 $T(M|\alpha\rightarrow\beta)$进行分解,首先分别确定活动$\alpha$和活动$\beta$的存在性及其所处的位置,继而在限定的范围内验证过程性质.该验证算法如下:

性质验证表达式 $T(M|\alpha\rightarrow\beta)=E(\alpha\in M)\wedge E(\beta\in M)\wedge X$ 中,若 $E(\alpha\in M)\wedge E(\beta\in M)$逻辑值为真,则进一步按如下算法构造表达式 X:

① 活动α和β属于相同的软件过程组件,即 $E_L(\alpha\in M)=E_L(\beta\in M)=C$,则 $X=T(C|\alpha\rightarrow\beta)$.该性质可以直接使用模型检测工具进行验证,而不需要进一步分解;返回算式 X 的检测结果,算法结束;

② 活动α和β分别属于不同的软件过程组件,$C=E_L(\alpha\in M)\neq E_L(\beta\in M)=C'$,则 $X=T(M|t_C\rightarrow\beta)\vee T(M|\alpha\rightarrow t_{C'})\vee T(M|t_C\rightarrow t_{C'})$,其中,$t_C=\{c|\ominus,c/C\}$,$t_{C'}=\{c'|\ominus,c'/C'\}$.

③ 若活动α不属于任何软件过程组件,即 $E_L(\alpha\in M)=act$,则考察添加活动 act 时,与之相邻的同层次活动α_M与活动β的时序关系,$X=T(M|a_M\rightarrow\beta)$;类似地,若活动$\beta$不属于任何软件过程组件,即 $E_L(\beta\in M)=act$,则考察

添加活动 *act* 时,与之相邻的同层次活动α_M与活动α的时序关系,$X=T(M|\alpha\rightarrow a_M)$.

若 $E(\alpha\in M)$及 $E(\beta\in M)$中至少有一个结果为假,即α,β中至少有一个活动在模型 *M* 中不存在,则无需构造表达式 *X*,直接给出结果 $T(M|\alpha\rightarrow\beta)$=False,即该性质在模型 *M* 中不成立;否则,使用模型检测工具递归地验证利用组件定义化简的过程性质 *X*,若检测结果为真,说明性质 $T(M|\alpha\rightarrow\beta)$成立,否则,性质不成立.这种验证方法以多次小规模模型性质验证为代价,降低了要验证的过程模型的规模,在大多数情况下,能够减少时间和空间开销,提高模型检测的效率.

4 应用示例

为验证 CSPM 的实用性,我们在中国科学院软件研究所的某软件开发项目中使用了本过程建模方法,并且进一步基于利用本方法所建立的软件过程模型的形式化特征对软件过程模型进行分析验证,以确保所建立的软件过程模型的正确性.

中国科学院软件研究所是一个达到 CMMI ML4 的软件组织,按组织内部规范,在项目执行前期要求对软件过程进行建模.本项目采用瀑布模型软件过程进行开发,大致可分为需求、设计、编码、测试这 4 个主要阶段,因此,首先确定使用瀑布模型组件 *WTF* 作为基础过程组件.根据组织要求,达到一定规模的开发项目需要在实际开发之前建立项目计划,因此需要在项目需求活动之前添加项目计划活动.根据软件过程组件化建模法则,需要执行操作$WTF_1=WTF+v/\overline{pla/req}$,向软件过程组件 *WTF* 的内部行为中添加活动节点 *pla*.接下来,根据项目具体情况和需要,逐个逐层地扩展过程组件中的具体活动:我们使用计划活动执行过程组件 *PLA* 扩展计划活动 *pla*,执行操作 $WTF_2=PLA\otimes WTF_1/pla$;继而使用执行需求活动的过程组件 *REQ*,执行设计活动的过程组件 *DES*,执行编码活动的过程组件 *COD*,以及执行测试活动的过程组件 *TES* 分别扩展组件 WTF_2 中的相应活动.进一步地,我们还分别使用过程组件 *A&D* 和 *SIM* 对设计活动过程组件 *DES* 中的“分析和设计(*ad*)”活动和编码活动过程组件 *COD* 中的“系统实现(*sim*)”活动进行扩展,得到最终的过程模型.因篇幅所限,上文中涉及的 *REQ* 到 *SIM* 的内部行为描述从略.最终得到的软件过程模型如图 8 所示.

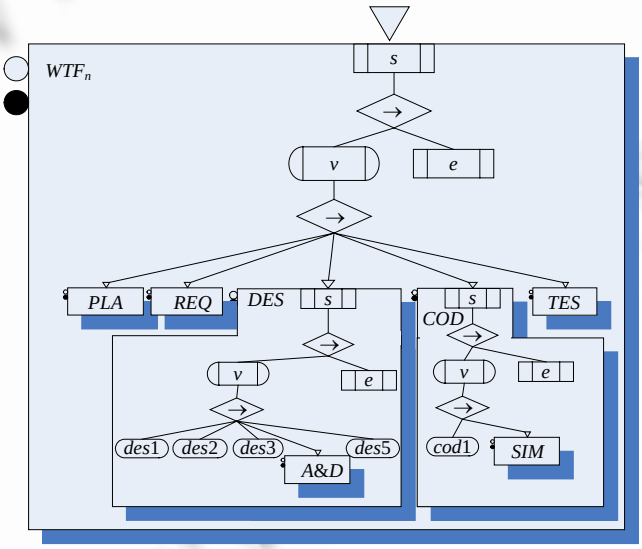


Fig.8 Constructed software process model

图 8 最终建立的软件过程模型

基于 CSPM 的过程组件操作表示法,上述建模过程可用如下操作表达式(14)表示:

$$WTF_n = SIM \otimes (A \& D \otimes (TES \otimes (COD \otimes (DES \otimes (REQ \otimes (PLA \otimes (WTF + v/\overline{pla/req})/pla)/req)/des)/cod)/tes)/ad)/sim \tag{14}$$

为简化表示,操作表达式(14)可以进一步变形为操作序列(15)的形式:

$$(\phi, WTF), (+, v/\overline{pla/req}), (\otimes, PLA/pla), (\otimes, REQ/req), (\otimes, DES/des), (\otimes, COD/cod), (\otimes, TES/tes), (\otimes, A \& D/a \& d), (\otimes, SIM/sim) \tag{15}$$

这样,软件过程组件化建模的全过程被严格记录,表达式所表示的软件过程模型唯一确定.同时,依照建模操作表达式或建模操作序列,该建模过程能够被严格地重复和回溯.

进一步地,我们利用组件化软件过程模型的组件化建模序列验证过程模型性质.下面举例说明验证过程模型中设计评审活动($des5$)的存在性 $E(des5 \in WTF_n)$以及活动时序性质:设计评审活动结束之后开始编码准备活动 $T(WTF_n|des5 \rightarrow cod1)$.首先,根据过程建模序列和活动存在性质的组件化验证方法,活动存在性质可以表示为性质验证表达式(16)的形式:

$$\begin{aligned} E(des5 \in WTF_n) = {} & E(des5 \in WTF) \vee E(des5 = pla) \vee (E(des5 \in PLA) \wedge \neg E(des5 = pla)) \vee \\ & (E(des5 \in REQ) \wedge \neg E(des5 = req)) \vee (E(des5 \in DES) \wedge \neg E(des5 = des)) \vee \\ & (E(des5 \in COD) \wedge \neg E(des5 = cod)) \vee (E(des5 \in A \& D) \wedge \neg E(des5 = a \& d)) \vee \\ & (E(des5 \in SIM) \wedge \neg E(des5 = sim)) \end{aligned} \tag{16}$$

用模型检测工具逐项验证,性质表达式(16)中分项 $E(des5 \in DES) \wedge \neg E(des5 = des)$的逻辑值为真,因此 $E(des5 \in WTF_n)$的值为真,即活动 $des5$ 在软件过程模型 WTF_n 中存在.

对于活动时序性质 $T(WTF_n|des5 \rightarrow cod1)$,根据组件化过程性质验证表达式的构造方法,该性质可以用性质验证表达式(17)表示:

$$T(WTF_n|des5 \rightarrow cod1) = E(des5 \in WTF_n) \wedge E(cod1 \in WTF_n) \wedge X \tag{17}$$

由于 $E(des5 \in WTF_n) \wedge E(cod1 \in WTF_n)$逻辑值为真,因此需要进一步构造表达式 X.由于 $DES = E_L(des5 \in WTF_n) \neq E_L(cod1 \in WTF_n) = COD$,因此,表达式(17)中的分项 X 有表达式(18)的形式:

$$X = T(WTF_n|t_{DES} \rightarrow cod1) \vee T(WTF_n|des5 \rightarrow t_{COD}) \vee T(WTF_n|t_{DES} \rightarrow t_{COD}) \tag{18}$$

其中,$t_{DES} = \{c|(\ominus, c/DES)\}$,$t_{COD} = \{c|(\ominus, c/COD)\}$.对表达式(18)递归地使用活动时序性质验证方法,根据 $E_L(t_{DES} \in WTF_n) = E_L(t_{COD} \in WTF_n) = WTF$,用模型检测工具直接验证过程活动时序性质 $T(WTF_n|t_{DES} \rightarrow t_{COD})$,得到结果逻辑值为真.因此,活动时序性质 $T(WTF_n|des5 \rightarrow cod1)$成立.即在过程模型 WTF_n 中,设计评审活动($des5$)结束之后才开始编码准备活动($cod1$).

5 结束语

本文提出了一种形式化的组件化软件过程建模方法 CSPM.该方法重点解决了软件过程重用和软件过程建模中的重用软件过程片段形式不统一、利用可重用软件过程片段建模方法不严格的问题.基于本方法,软件过程组件可以作为软件过程重用的标准形式,有利于人们的交流和相互理解.同时,在本方法中还形式化定义了使用软件过程组件建立软件过程模型的操作法则.基于这些法则,一个软件过程的建模过程能够被严格地描述、记录、重复和回溯,这对于软件过程建模过程的重用及软件过程的改进有着积极的作用.除此以外,CSPM 还可以针对某些特定性质对组装后的软件过程模型进行验证,并可把对该模型的验证问题转化为对其对应组件的一系列子验证问题,从而指数地减少需要搜索的状态空间,使原来在某些特定环境下代价过大的性质验证问题简化成验证代价较小的一系列问题.最后,本文还给出了利用本方法建立的软件过程模型的应用实例,这进一步表明了本建模方法的实用性.

作为今后进一步的工作,我们首先计划建立利用 CSPM 描述的软件过程组件库,并在实践中不断改进,以给出一套能够覆盖绝大多数常见软件过程模型的过程组件体系,并且建立过程组件间的关系、约束等限制条件,进一步规范建模过程;第二,需要以过程组件库、过程建模工具、模型检测工具等为基础,建立集成的软件过程建模支撑环境,并使之整合到现有的 PSEE 系统当中,在实践中验证和改进本建模方法;第三,进一步深化对本方

法,特别是加强对使用本方法建立的软件过程模型的分析和验证工作的研究,建立适用于更一般的软件过程性质的软件过程性质组件化验证方法,从而进一步支持可信软件过程的建模,达到生产可信软件的目的.

References:

[1] Hollenbach C, Frakes W. Software process reuse in an industrial setting. In: Sitaraman M, ed. Proc. of the 4th Int'l Conf. on Software Reuse (ICSR). Washington: IEEE Computer Society, 1996. 22−30. http://portal.acm.org/citation.cfm?id=853482 [doi: 10.1109/ICSR.1996. 496110]

[2] Reis RQ, Reis CAL, Nunes DJ. Automated support for software process reuse: Requirements and early experiences with the APSEE model. In: Proc. of the 7th Int'l Workshop on Groupware (CRIGW 2001). Los Alamitos: IEEE Computer Society, 2001. 50−55. http://www.computer.org/portal/web/csdl/doi/10.1109/CRIWG.2001.951766 [doi: 10.1109/CRIWG.2001.951766]

[3] Ambler SW. Process Patterns: Building Large-Scale Systems Using Object Technology. New York: SIGS Books/Cambridge University Press, 1998.

[4] Hagen M, Gruhn V. Process patterns—A means to describe processes in a flexible way. IEE Digest, 2004,2004(911):32−39. [doI: 10.1049/ic:20040441]

[5] Hagen M, Gruhn V. Towards flexible software processes by using process patterns. In: Hamza MH, ed. Proc. of the IASTED Conf. on Software Engineering and Applications (SEA 2004). MIT Cambridge: IASTED/ACTA Press, 2004. 436−441.

[6] Milner R, Parrow J, Walker D. A calculus of mobile processes I. Journal of Information and Computation, 1992,100(1):1−77. [doi: 10.1016/0890-5401(92)90008-4]

[7] Li MS. Assessing 3-D integrated software development processes: A new benchmark. In: Wang Q, *et al*, eds. Proc. of the Int'l Software Process Workshop and Int'l Workshop on Software Process Simulation and Modeling (SPW/ProSim 2006). LNCS 3966, Berlin: Springer-Verlag, 2006. 15−38.

[8] Osterweil LJ. Software processes are software too. In: Riddle WE, ed. Proc. of the 9th Int'l Conf. on Software Engineering. New York: ACM Press, 1987. 2−13.

[9] Osterweil LJ. Software processes are software too, revisited: An invited talk on the most influential paper of ICSE 9. In: Conradi R, ed. Proc. of the 19th Int'l Conf. on Software Engineering (ICSE'97). Berlin: Springer-Verlag, 1997. 540−548. [doi: 10.1109/ICSE.1997.610401]

[10] Lonchamp J. A structured conceptual and terminological framework for software process engineering. In: Osterweil LJ, ed. Proc. of the 2nd Int'l Conf. on the Software Process. Berlin: Springer-Verlag, 1993. 41−53. [doi: 10.1109/SPCON.1993.236823]

[11] Li MS, Yang QS, Zhai J. Systematic review of software process modeling and analysis. Journal of Software, 2009,20(3):524−545 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/20/3432.htm [doi: 10.3724/SP.J.1001.2009.03432]

[12] Perry DE. Practical issues in process reuse. In: Proc. of the Int'l Software Process Workshop. Los Alamitos: IEEE Computer Society, 1996. 12−14. [doi: 10.1109/ISPW.1996.654357]

[13] Ambler SW. More Process Patterns: Delivering Large-Scale Systems Using Object Technology. New York: Cambridge University Press, 1999.

[14] van der Aalst WMP, ter Hofstede AHM, Kiepuszewski B, Barros AP. Workflow patterns. Distributed and Parallel Databases, 2003, 14(3):5−51. [doi: http://dx.doi.org/10.1023/A:1022883727209]

[15] Russell N, ter Hofstede AHM, Edmond D, van der Aalst WMP. Workflow data patterns. QUT Technical Report, FIT-TR-2004-01, Brisbane: Queensland University of Technology, 2004.

[16] Russell N, ter Hofstede AHM, Edmond D, van der Aalst WMP. Workflow resource patterns. BETA Working Paper Series, WP 127. Eindhoven: Eindhoven University of Technology, 2004.

[17] Russell N, van der Aalst WMP, ter Hofstede AHM. Workflow exception patterns. In: Dubois E, Pohl K, eds. Proc. of the 18th Int'l Conf. on Advanced Information Systems Engineering (CAiSE 2006). LNCS 4001, Berlin: Springer-Verlag, 2006. 288−302. [doi: 10.1007/11767138_20]

[18] Riehle D, Zullighoven H. Understanding and using patterns in software development. Theory and Practice of Object Systems, 1996, 2(1):3−13. [doi: 10.1002/(SICI)1096-9942(1996)2:1%3C3::AID-TAPO1%3E3.0.CO;2-%23]

[19] Li MS. Expanding the horizons of software development processes: A 3-D integrated methodology. In: Li MS, *et al*., eds. Proc. of the Int'l Software Process Workshop (SPW 2005). LNCS 3840, Berlin: Springer-Verlag, 2005. 54–67.

[20] Yang QS, Li MS, Wang Q, Yang G, Zhai J, Li J, Hou L, Yang Y. An algebraic approach for managing inconsistencies in software processes. In: Wang Q, *et al*., eds. Proc. of the Int'l Conf. on Software Process (ICSP 2007). LNCS 4470, Berlin: Springer-Verlag, 2007. 121–133. [doi: 10.1007/978-3-540-72426-1_11]

[21] Cleaveland R, Li T, Sims S. The concurrency workbench of the new century: User's manual. SUNY at Stony Brook, 2000.

[22] Holzmann GJ. The model checker spin. IEEE Trans. on Software Engineering (TSE), 1997,23(5):279–295. [doi: 10.1109/32.588521]

[23] The mobility workbench. 1999. http://www.it.uu.se/research/group/mobility/mwb

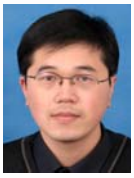
[24] Bradfield J, Stirling C. Modal logics and μ-calculi: An introduction. In: Bergstra J, Ponse A, Smolka S, eds. Handbook of Process Algebra. New York: Elsevier, 2001. 293–330.

[25] Emerson EA. The beginning of model checking: A personal perspective. In: Orna G, Helmut V, eds. Proc. of the 25 Years of Model Checking: History, Achievements, Perspectives. LNCS 5000, Berlin: Springer-Verlag, 2008. 27–45. [doi: 10.1007/978-3-540-69850-0_2]

附中文参考文献:

[11] 李明树,杨秋松,翟健.软件过程建模方法研究.软件学报,2009,20(3):524–545. http://www.jos.org.cn/1000-9825/20/3432.htm [doi: 10.3724/SP.J.1001.2009.03432]

翟健(1981－),男,北京人,博士生,主要研究领域为软件过程方法与技术.



肖俊超(1978－),男,博士,助理研究员,CCF 会员,主要研究领域为软件过程技术.

杨秋松(1977－),男,博士,助理研究员,CCF 会员,主要研究领域为软件过程方法与技术.

李明树(1966－),男,博士,研究员,博士生导师,CCF 高级会员,主要研究领域为软件过程方法与技术.