第 21 卷第 2 期
2 0 1 5 年 2 月

计算机集成制造系统
Computer Integrated Manufacturing Systems

Vol. 21 No. 2
Feb. 2 0 1 5

DOI:10.13196/j.cims.2015.02.003

CP-PMTT:一个基于控制流模式的过程模型转换工具

邵真禄,张　莉+,凌济民
(北京航空航天大学 计算机学院,北京　100191)

摘　要:目前大多数过程模型转换方法采用基于元类映射的转换规则,鉴于因不同过程建模语言在建模符号和语法约束上的差异,而使基于元类映射的转换规则在应对过程模型转换时存在明显局限,提出一种基于控制流模式的过程模型转换框架,转换框架包括一个支持用户自定义控制流模式的控制流模式定义框架,控制流模式定义框架是转换框架的转换核心。通过建立源语言、目标语言和转换核心的映射关系,生成源语言到目标语言的转换规则。基于上述框架实现了一个过程模型转换工具。通过业务流程建模标记到另一种工作流语言的转换场景,对转换工具进行了验证。

关键词:过程模型;模型转换;控制流模式;转换规则
中图分类号:TP311　　**文献标识码**:A

CP-PMTT:a control-flow pattern based process model transformation tool

SHAO Zhen-lu, ZHANG Li+, LING Ji-min
(Department of Computer Science and Engineering, Beihang University, Beijing 100191, China)

Abstract: The transformation rules based on meta-class mapping were used in most existing process transformation approaches, which was limited to deal with process model transformation for huge differences between process modeling languages in modeling notation and syntactic. To solve this problem, a control-flow pattern based transformation framework was proposed, which included a framework of control-flow pattern definition as the transformation core. The transformation rule from source language to target language could be auto generated by establishing the mapping relationship with transformation core respectively. A process model transformation tool was developed based on the transformation framework, and the transformation from Business Process Modeling Notation (BPMN) to Yet Another Workflow Language (YAWL) was used to validate this transformation tool.

Key words: process model; model transformation; control-flow pattern; transformation rule

0　引言

随着企业对过程重视程度的不断提高,过程模型在企业中变得越来越重要,作为企业过程管理中的一项关键技术,过程模型转换正受到越来越多的关注。通常所说的过程模型转换是指将以某种过程建模语言建立的过程模型转换成使用另外一种过程建模语言建立的模型,并尽可能保持源模型和目标模型语义的一致。

所谓过程是将“输入转换为输出的相互关联和相互作用的一组活动”[1],活动表示过程中需要完成的任务,活动之间的关系则决定了活动的执行顺序,本文将活动与其之间的关系称为过程控制流,由于通常认为控制流是过程的核心,很多过程模型的转换工作专注于过程控制流的转换[2]。本文将通过删除过程模型中的控制流无关信息(资源、中间产品等)得到的模型称为过程控制流模型,同其他的过程模型转换工作类似,本文的过程模型转换专注于过

收稿日期:2014-12-01。Received 01 Dec. 2014.
基金项目:国家自然科学基金资助项目(61170087)。**Founadation item**: Project supported by the National Natural Science Foundation, China (No. 61170087).

程控制流模型之间的转换。

为了实现过程模型之间的转换,通常采用的方法是建立源语言和目标语言之间基于元类的映射关系[2],根据元类映射关系制定转换规则,但是这种基于元类映射的转换方法存在如下局限性:

(1)*不同的建模语言相对应的建模符号表示的控制流结构不尽相同*　例如在业务流程建模标记(Business Process Modeling Notation, BPMN)中可以使用"过程"包含两个不相关"任务"的形式表示并行分支结构[3],而在另一种工作流语言(Yet Another Workflow Language, YAWL)中,与 BPMN 的"过程"相对应的"组合任务"则无法表示并行分支结构[4]。此时采用元类映射的方法将无法得到正确的转换结果。

(2)*控制流结构在不同建模语言中实现的粒度不同*　例如在 YAWL 中可以使用"多实例原子任务"表示多实例控制流结构[4],而在 BPMN 中没有与 YAWL 的"多实例原子任务"相对应的建模元素,只能采用组合多种建模元素的形式表示多实例控制流结构[3]。此时采用元类映射的方法将无法得到转换结果。

对于上述问题,一种简单的解决方案是采用元类映射加特定控制流片段映射的方法来制定转换规则[5],但是这种方法要求针对每一种转换场景都需要人工制定源语言到目标语言特定片段之间的映射规则,因此不具有通用性,也增加了模型转换中人员的负担。

对此,本文提出一种基于控制流模式进行过程模型转换的思想,所谓控制流模式是对在不同过程模型中出现的典型控制流语义单元的抽象[6],控制流模式在过程模型中的实现称为一个控制流模式实例。如果定义的控制流模式的集合的描述能力是足够的,则一个过程模型总可以被拆分成一组控制流模式实例,而不剩下任何元素(本文假设过程模型中没有孤立任务存在,过程模型可以看做一个强连通图)。因此通过制定基于控制流模式的转换规则,可将源模型中的控制流模式实例转换到目标模型中的控制流模式实例,进而实现源模型到目标模型的转换。

基于上述思想,本文提出一种基于控制流模式的过程模型转换框架,并基于该框架实现了一个基于控制流模式的过程模型转换工具(Control-flow Pattern based Process Model Transformation Tool,CP-PMTT),将过程模型转换的粒度由元类提高到控制流模式,解决了基于元类转换的局限性问题,进而提高过程模型转换的正确率,避免有些控制流结构无法得到转换结果。

1　问题分析

假设:$Model_1$ 表示过程模型转换中的源模型,$Model_2$ 表示目标模型;建模 $Model_1$ 所使用的过程建模语言为 PML_1,建模 $Model_1$ 使用的过程建模语言为 PML_2;Rule 表示 PML_1 到 PML_2 的转换规则;P_{Rule} 表示采用某种编程语言实现了转换规则 Rule 的过程模型转换程序。因此从 $Model_1$ 到 $Model_2$ 的转换可以描述为

$$Model_2 = P_{Rule}(Model_1)。$$

假设 $MPML_1$ 和 $MPML_2$ 分别表示 PML_1 和 PML_2 中元类的集合,则目前常用的基于元类的过程模型转换规则 MRule 可以表示为

$$MPML_2 = MRule(MPML_1)。$$

由于不同过程建模语言在建模符号和语法约束上的差异,各种建模语言在过程控制流模式实现方式上也各不相同。假设 GPR_1 和 GPR_2 分别表示过程建模语言 PML_1 和 PML_2 中控制流模式实例的集合,则由于前文分析的原因,在采用转换规则 MRule 时,有可能存在 $PaS_1 \in GPR_1$,使得在转换的过程中找不到 $PaS_2 \in GPR_2$ 与其对应,造成源模型和目标模型控制流结构不一致。

本文提出一种基于控制流模式的过程模型转换框架,该框架本质上可以看作是为模型转换提供一种新的基于控制流模式的过程模型转换规则 PRule,基于这种规则的过程模型转换可以表示为

$$GPR_2 = PRule(GPR_1);$$
$$Model_2 = P_{PRule}(Model_1)。$$

通过上面的公式可以看出,要完成基于控制流模式的过程模型转换,需要关注下面几方面问题:GPR_1 和 GPR_2 集合中需要包含多少种控制流模式的实现,如何描述控制流模式在各种建模语言中的实现形式,以及如何根据控制流模式在源模型、目标模型中的模式实现形式的集合建立基于控制流模式的过程模型转换规则。

基于上述问题,本文提出如图 1 所示的过程模型转换框架。该框架包括控制流模式层、转换规则层和执行层三个层次。最底层是控制流模式层,包括一个作为转换核心的控制流模式定义框架。由于

控制流模式代表过程模型中的典型控制流，该定义框架应该能够定义具备足够表达能力的控制流模式集合。控制流模式定义框架是该过程模型转换框架的基础。框架的中间层是转换规则层，在该层本文设计并实现了一种基于控制流模式的过程模型转换描述语言，使用该语言来描述转换核心中的控制流模式在源语言、目标语言中的模式实现，建立源语言、目标语言和转换核心的映射关系，基于这些映射关系可以生成源语言到目标语言的转换规则。框架的最上层是执行层，该层主要通过转换工具执行生成的转换规则，将源模型转换到目标模型。

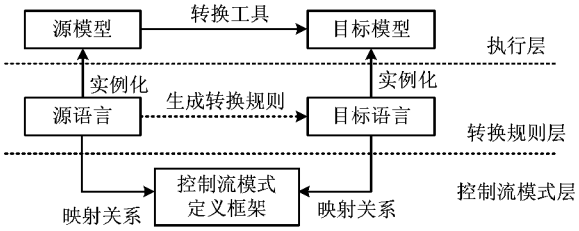


图1 过程模型转换框架

2 控制流模式定义框架

首先提出过程控制流的相关概念，然后定义过程控制流中的原子构造算子。基于这些原子构造算子定义 8 种原子控制流模式，基于这些原子控制流模式和组合的实现关系可以定义组合控制流模式，进而实现一个支持用户自定义控制流模式的控制流模式定义框架。最后对该控制流模式定义框架的完备性进行讨论。

2.1 过程控制流相关概念

通过对 BPMN[3]、开源工作流管理 jBPM 中的建模语言(JBoss jBPM Process Definition Language，JPDL)[7]、可视化过程建模语言(Visual Process Modeling Language，VPML)[8]、YAWL[4] 4 种典型的过程建模语言进行分析，发现控制流主要由过程中的任务及其之间的相互作用、相互关联决定。过程控制流相关概念可以表示为一个四元组：ProcessControlflowRelatedConcepts={S,E,T,R}，其中：

S 表示过程的开始(Start)，只有过程开始后，过程中的活动才能获得执行权力并执行；

E 表示过程的结束(End)，当过程结束后，过程中的所有活动都将停止执行；

$T=\{T_1,T_2,T_3,\cdots,T_n\}$($n\geqslant 1$)表示过程中所包含的任务的集合，任务表示过程执行中特定的步骤，且应该具有一系列属性，如可执行性、可触发性和可取消性等；

$R=\{r_1,r_2,r_3,\cdots,r_n\}$是过程的开始、任务、过程的结束间关系的集合，其中 r_i 是定义在 $t_k\times t_l$ 上的关系，$t_k\in T\cup S$，$t_l\in T\cup E$，$R\subset(T\cup S)\times(T\cup E)$，任务间的关系用来定义不同任务间的执行顺序和制约关系。

2.2 原子构造算子

由于过程中任务间的关联关系多种多样，无法为每一种关系提供一种专门的建模符号，本文通过分析过程建模语言归纳出如下一组原子构造算子，这些构造算子虽然只能直接描述有限的控制流，但是通过组合不同任务和这些构造算子可以表达更复杂的控制流。

(1)顺序流(sequence-flow)，用来连接两个任务，表示它们之间顺序执行。

(2)取消流(cancel-flow)，用来连接两个任务，表示取消它们中的源任务而执行目标任务。

(3)并行分支器(and-splitter)，用来表示一条控制流分裂为两条或两条以上、并行执行的控制流分支。

(4)异或选择分支器(xor-splitter)，用来表示一条控制流分裂为两条或两条以上的、选择执行的控制流分支。异或选择分支器的后续控制流只有一条能够获得执行权限。

(5)同步合并器(and-joiner)，用来表示两条或两条以上控制流分支合并为一条控制流。只有当所有的分支都执行完成后，同步合并器之后的控制流分支才能获得执行权限。

(6)异或合并器(xor-joiner)，用来表示两条或两条以上控制流分支合并为一条控制流。每当异或合并器前驱分支中的一条执行完成，异或合并器的后续控制流就会获得执行权限。

(7)或合并器(or-joiner)，用来表示两条或两条以上控制流分支合并为一条控制流。这种合并器会智能地判断其前驱分支的执行情况，当仍有前驱分支可能执行时，它将等待这些前驱分支；当有前驱分支执行完成，除此之外的所有前驱都不可能再执行完成时，其后续控制流才会获得执行权限。

(8)鉴别合并器(discriminate-joiner)，用来表示两条或两条以上控制流分支合并为一条控制流。当鉴别合并器前驱分支中的一条执行完成后，其后续

控制流就会获得执行权限。如果其他前驱分支也执行完成,则鉴别合并器会将其忽略。

2.3　控制流模式定义框架

目前已经有很多针对控制流模式的研究,例如荷兰学者 Aalst 于 2003 年提出的 20 种控制流模式[6]、Murzek 提出的结构模式[9]等。目前的研究大多试图穷举已知的控制流模式[6,9-11],但是穷举并不是研究一个事物本质的合理方法。因此,本文提出一个基于原子控制流模式加组合实现关系的控制流模式定义框架,如图 2 所示。

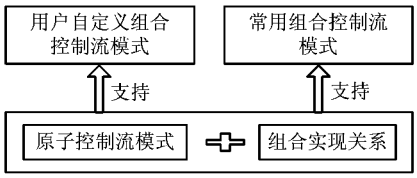


图2　控制流模式定义框架

该框架的核心是:首先定义控制流模式中的一组原子控制流模式,然后基于这些原子控制流模式和组合实现关系,定义已知的常见的控制流模式。原子控制流模式指不再包含其他控制流模式的控制流模式。控制流模式定义框架中的组合实现关系指通过组合任务和与原子控制流模式对应的原子构造算子,实现比原子控制流模式更为复杂的控制流模式。这些控制流模式概念在控制流模式定义框架中被称为组合控制流模式。该控制流模式定义框架是可扩展的,当需要扩展控制流模式集合时,即使新的控制流模式不能由原有的原子控制流模式加组合实现关系定义,也可以通过添加原子控制流模式,使控制流模式定义框架支持该控制流模式。

基于上述原子构造算子,本文给出控制流模式定义框架中的原子控制流模式集合。控制流模式定义框架中的原子控制流模式集合由 8 种原子控制流模式组成:AtomicPatterns={sequence, cancel, and-split, xor-split, and-join, xor-join, or-join, discriminate-join}。其中:

(1)squence 表示顺序模式,指过程中的活动按照预先设定的顺序一个接一个地执行。顺序模式由实现层抽象中的一个二元算子顺序流和表达前驱任务、后继任务的两个原子任务构成。

(2)cancel 表示取消模式,指一个可执行或正在执行的活动被强制取消了。取消模式由实现层抽象中的一个二元算子取消流和表示取消源任务、取消目标任务的两个原子任务构成。

(3)and-split 表示并行分支模式,指流程在某个活动之后产生多个分支,这些分支同时运行。并行分支模式由实现层抽象中的一个多元算子并行分支器、一个表示前驱控制流的顺序流和多个表示后继控制流的顺序流构成。

(4)xor-split 表示异或选择分支模式,指从多个可选分支中选择一个执行。异或选择分支模式由实现层抽象中的一个多元算子异或选择分支器、一个表示前驱控制流的顺序流和多个表示后继控制流的顺序流构成。

(5)and-join 表示并行合并模式,指多个并行的分支合并成一个分支,必须等到所有分支都执行完后,才能激活后续活动。并行合并模式由实现层抽象中的一个多元算子并行合并器、多个表示前驱控制流的顺序流和一个表示后继控制流的顺序流构成。

(6)xor-join 表示异或合并模式,指两个或多个可选分支(不允许任意两个或多个分支并行)在某一点处合并成一个分支。异或合并模式由实现层抽象中的一个多元算子异或合并器、多个表示前驱控制流的顺序流和一个表示后继控制流的顺序流构成。

(7)or-join 表示或合并模式,指多条分支汇合到一条分支上,如果多个分支中被采取的分支可能超过一条,则将其同步,同步完这些分支后,后续分支开始执行。或合并模式由实现层抽象中的一个多元算子或合并器、多个表示前驱控制流的顺序流和一个表示后续控制流的顺序流构成。

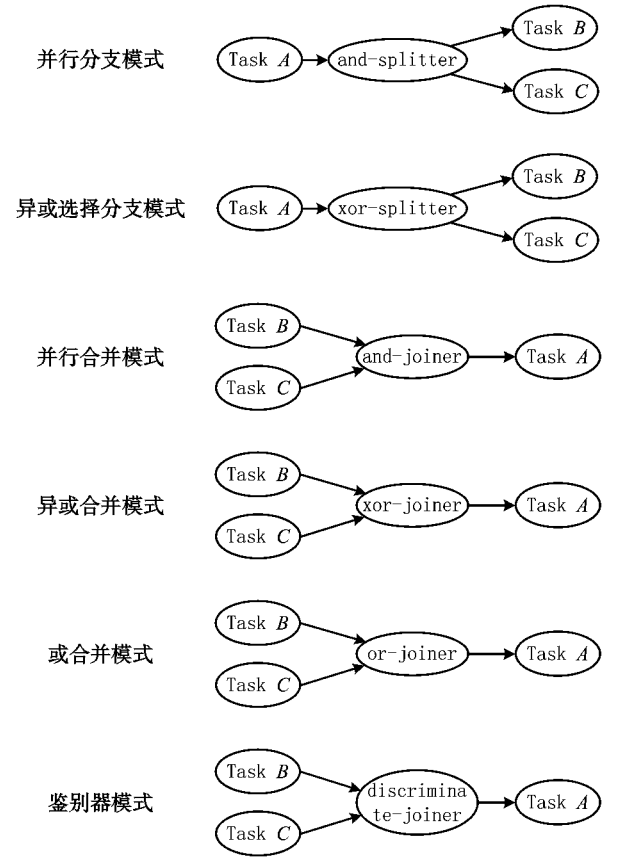
(8)discriminate-join 表示鉴别器模式,指等待前驱分支之一完成以激活后续分支,等待剩余分支并"忽略"它们。鉴别器模式由实现层抽象中的一个多元算子鉴别合并器、多个表示前驱控制流的顺序流和一个表示后续控制流的顺序流构成。

这 8 种原子控制流模式的图形化表示如表 1 所示。

表 1　原子控制流模式的图形化表示

原子控制流模式	图形化表示
顺序模式	Task *A* → Task *B*
取消模式	Task *A* → Task *B*（cancel）

续表 1



除了上述 8 种原子控制流模式，该框架还支持使用这些原子控制流模式和原子构造算子相互组合的方式来构造控制流模式。如图 3 所示，任意循环模式可以由任务、顺利流、异或分支器和异或合并器共同构成。

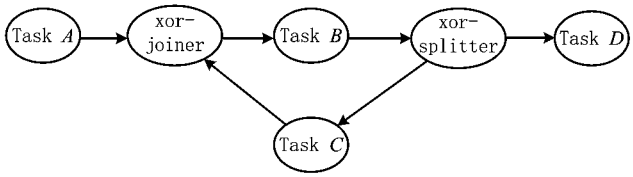


图3 循环模式图形化表示

2.4 完备性说明

为了验证控制流模式定义框架的建模能力，本文使用该框架对 20 种工作流模式进行建模。在已有的过程控制流模式相关研究中，荷兰学者 Aalst 等提出的 20 种工作流模式[6]最具影响。实践证明，20 种工作流模式可以使用 8 种原子模式和组合实现关系描述。20 种工作流模式中有 8 种与本文定义的原子模式在语义上一致，另外 12 种则可基于这 8 种原子模式组合实现。因此，本文方法在描述常见控制流模式方面是充分可行的。

应当指出，上述 8 种原子模式只是一些最基本的控制流概念单元，它们只能组合实现一些在不同过程建模语言中经常使用的控制流模式，从这个意义上讲，本文提出的原子控制流模式是不完备的。但是该控制流模式定义框架具备良好的可扩展性，这是因为在提供基于原子模式和组合实现关系的控制流模式定义框架的同时，即使新的控制流模式不能由原有的原子控制流模式加组合实现关系定义，也可以通过添加原子控制流模式，使控制流模式定义框架支持该控制流模式，这种方式可以从整体上提高该框架对控制流模式的表现能力。

3 基于控制流模式的过程模型转换描述语言

首先对控制流模式在过程建模语言中的实现形式进行分析。为了描述不同建模语言中控制流模式的实现形式，本文提出一种工作流结构图来描述过程模型中的控制流结构，然后使用环路等价算法对控制流结构图进行规约，规约后的控制流结构图由单入口单出口块和节点构成；接着在规约后的工作流结构图的基础上，设计并实现了过程模型转换描述语言（Process Model Transformation Lanuage，PMTL）；最后介绍根据源语言、目标语言和转换核心的映射关系生成源语言到目标语言转换规则的策略。

3.1 控制流模式实现

控制流模式实现指控制流模式在过程建模语言中的具体实现，在过程建模语言中由活动和构造算子组成。控制流模式定义框架中包括原子控制流模式和组合控制流模式，与这些相对应，在具体的过程建模语言中包括原子控制流模式实现和组合控制流模式实现。在实现组合控制流模式时，如果一种组合控制流模式可以由原子任务节点、表示顺序关系的原子建模符号和对应于该控制流模式的原子建模符号实现，则这种实现称为该组合模式在该过程建模语言的原子实现，例如在 YAWL 语言中可以使用“多实例任务”建模符号表示确定个数的多实例模式[4]；而如果一种组合控制流模式依靠组合原子建模符号实现的方式来实现，则称其为该组合模式在过程建模语言的组合实现，例如在 YAWL 语言中还可以使用任务和并行分支器、并行合并器组合的方式来实现[4]。综上所述，在过程建模语言中，转换核心中控制流模式的实现主要包括原子控制流模式

实现、组合控制流模式的原子实现和组合控制流模式的组合实现三种。

3.2　控制流结构图

为了描述不同的过程建模语言中的控制流模式实现,需要抽象出过程模型中独立于建模语言的控制流结构。文献[12]提出采用工作流图描述和建模语言无关的常见控制流信息。本文描述的控制流模式实现与具体的过程建模语言相关,因此对工作流图进行了改造,为工作流图中的节点添加了一个属性集合,用于保存与特定的过程建模语言相关的信息。改造后的工作流图(后文统称为工作流结构图)是一个有向的连通图 $G=(N,E)$。其中:

(1)N 表示模型中所有节点的集合。

(2)$E\subset N\times N$,E 为节点间有向边的集合,这些有向边表示控制流。

(3)对 $\forall n\in N$,n 都具有一个属性 attribute 集合,用来描述和特定的过程建模语言相关的信息,其中对 $\forall$ attr$\in$attribute,attr 的属性名称和属性值都由用户自定义完成。

以如图 4 所示的一个 YAWL 模型为例,过程从起始任务 Start 开始到达任务 g_1 后,并行执行 g_1 的两个后续分支。其中的一个分支串行地执行 a_1,a_2 和 a_3 这几个任务,另外一个分支则执行由 g_2,a_4 和 g_3 构成的循环。这两个分支在 g_4 汇合,最后到达终止节点 End,过程结束。该过程采用工作流结构图表示,如图 5 所示(只列出部分节点的属性集合),圆形图标表示过程模型中的节点元素,带有箭头的连接线表示控制流;Start 节点采用{type=start}属性集合来表示自身为 YAWL 模型中的开始节点;a_1 节点采用{type=task,join=none,split=none}属性集合来表示自身为 YAWL 模型中的原子任务;g_4 采用{type=task,join=and,split=none}属性集合来表示自身为 YAWL 模型中的与合并任务。

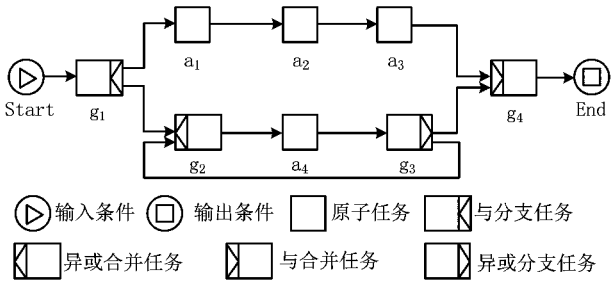


图4　YAWL模型实例

为便于描述过程模型中控制流模式的实现方式,本文通过环路等价算法[13]对工作流结构图进行处理,规约后的工作流结构图由规则单入口单出口过程块和节点构成。定义 1～定义 3 给出了规则单入口单出口过程块的相关定义(如无特殊说明,本文后续的过程块都指代为规则的单入口单出口过程块):

定义 1　令 x 和 y 为工作流结构图 G 中的两条边:如果每条从起始节点到 y 的通路上都包含 x,则称 x 控制 y;如果每条从 y 到终止节点的通路上都包含 x,则称 y 受控于 x。

定义 2　工作流结构图 G 中的单入口单出口过程块是一个由有序边对(m,n)定义的过程块,当且仅当:①m 和 n 是两条相异的边;②m 控制 n;③n 受控于 m;④每条包含 m 的回路中均包含 n,每条包含 n 的回路中均包含 m。

定义 3　单入口单出口过程块(a,b)是规则的,当且仅当:①对任何单入口单出口块(a,b'),b'控制 b;②对任何单入口单出口块(a',b),a 受控于 a'。

对图 5 进行规约,经过规约后的工作流结构图如图 6 所示(省略了节点属性集合),并用虚线框标出了每个过程块。对于给定过程块 F,称 F 的父过程块为包含 F 的最小过程块 F′,并称 F 为 F′的子过程块。将过程块的子过程块替换为一个活动节点,不会影响该过程块的正确性。以图 6 为例,过程块 B 是过程块 D,E 和 F 的父过程块,而过程块 H 是 C 的子过程块。在描述由 g_1,B 和 C 组成的并行分支控制流模式时,可以将 B 和 C 作为一个活动节点来考虑,而无需考虑其内部结构,从而简化对控制流模式实现的描述。

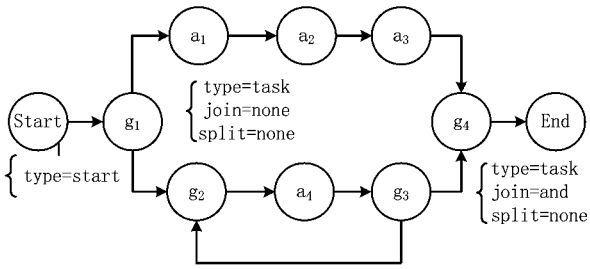


图5　控制流结构图实例

3.3　过程模型转换描述语言

基于上述工作流结构图,本文设计并实现了 PMTL。PMTL 通过描述控制流模式在过程建模语言中的表现形式来建立过程建模语言和转换核心之间的映射关系。PMTL 由属性定义、元素定义和

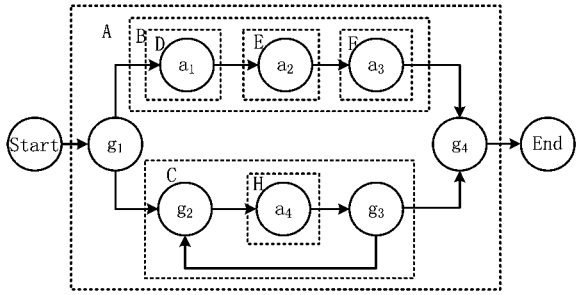


图6 规约后的工作流结构图

控制流模式实现定义三部分构成。

3.3.1 属性定义

属性定义部分主要用于描述工作流结构图中每个节点的属性，每一个属性都由属性名称和属性取值集合构成，属性名称和属性的取值集合由用户自定义。属性定义的使用实例如表 2 所示。

表 2 属性定义实例

YAWL 语言节点属性说明	属性定义
节点类型包括 input，output，condition，task，multTask	define type={input，output，condition，task，mulTask}
节点分支属性包括 and，or，xor，none	define split={and，join，xor，none}
节点合并属性包括 and，or，xor，none	define join={and，join，xor，none}

3.3.2 元素定义

元素定义部分通过定义节点中属性的取值来定义过程建模语言中的元素，一个节点必须满足一个元素定义下所有属性的取值，才能满足该元素的定义。元素定义的使用实例如表 3 所示。

表 3 元素定义实例

YAWL 建模元素	属性取值	元素定义

原子任务 	节点类型为 task
	节点分支属性为 none
	节点合并属性为 none
		define Task
		{
		type=task;
		join=none;
		split=none;
		}

与合并任务	节点类型为 task
	节点分支属性为 none
	节点合并属性为 and
		define And_Join_Task
		{
		type=task;
		join=and;
		split=none;
		}

3.3.3 控制流模式实现定义

控制流模式实现定义由谓词、规则和控制流模式三部分组成，下面分别对这三部分进行说明。

(1)模式定义的基本概念是谓词，根据谓词的目标类型可以分为节点谓词和块谓词，其中节点谓词应用于节点上的属性，块谓词应用于过程块中的节点。谓词的使用示例如表 4 所示。

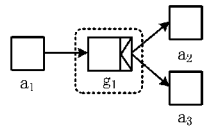
表 4 谓词使用实例

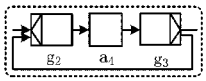
	谓词含义	谓词定义
节点谓词	节点包含一条以上的出边 节点为与合并任务元素	OutputEdgesCount>1 Element=And_Join_Task
块谓词	起始节点满足 Xor_Join 规则的过程块 终止节点满足 And_Split 规则的过程块	BeginWith=Xor_Join EndWith=And_Split

(2)比谓词高一级的概念是规则，规则由一系列谓词组成。根据规则目标类型分为节点规则和块规则，其中节点规则由若干属性谓词组成，用于描述过程模型中的模型节点；块规则由若干节点谓词组成，用于描述过程模型中的过程块。规则的使用示例如表 5 所示。

表 5 规则使用实例

目标控制流结构	规则定义

节点规则 并行分支 	rule And_Split on Node
	{
	Element=And_Split_Task;
	InputEdgesCount=1;
	OutputEdgesCount>1;
	}

块规则 循环 	rule Loop on Block
	{
	StartWith=Xor_Join;
	EndWith=Xor_Split;
	}

(3)比规则高一级的概念是控制流模式，由于控制流模式在过程建模语言中有多种实现形式，例如组合模式对应有组合模式的原子实现和组合模式的组合实现方式，因此一个控制流模式由一个到多个规则组成。对于指定的控制流模式，只要过程块或节点满足该控制流模式规则集合中的一条规则，就称该过程块或模型节点满足该控制流模式。其中控

制流模式中包含的节点规则对应控制流模式实现中的原子控制流模式实现和组合控制流模式的原子实现,块规则对应控制流模式实现中的组合控制流模式的组合实现。

表 6 给出了过程模型转换描述语言的扩展巴科斯范式(Extended Backus Naur Form, EBNF)定义,定义中的过程模型转换描述语言(PMTL_Description)由属性定义(attributes)、元素定义(elements)和控制流模式实现定义(patterns)三部分构成。控制流模式实现定义由一个到多个控制流定义(pattern)组成,控制流定义由一个到多个规则(rule)组成,规则由一个到多个谓词(predicate)组成。

表 6　PMTL 的 EBNF 定义

```
PMTL_Description=attributes,elements,patterns;
attributes="Attributes","{",{attribute},attribute,"}";
attribute=identifiter,"=","{",{identifier,","},identifier,"}",new_
line;
elements="Elements","{",{element},element,"}"
element="define",identifier,"{",{key_value},key_value,"}",new_
line;
key_value=identifier,"=",identifier,";",new_line;
patterns="Patterns","{",{pattern},pattern,"}";
pattern="pattern",identifier,":",{rule},rule;
rule="rule",identifier,"on","node"|"block","{",{predicate},predi-
cate,"}"
predicate="Element"|"InputEdgesCount"|"OutputEdgesCount"|"
BeginWith"|"Inner"|"EndWith",relation,";",identifier,new_line;
relation=">"|'='|'<'|'>='|'<=','! =';
identifier={alpha},alphanum;
alphanum=alpha|number;
alpha=? alphabetic characters?;
number=? decimal digits?;
new_line="\n";
```

3.4　转换规则生成策略

结合工作流结构图,过程模型转换描述语言通过属性定义、元素定义和控制流模式实现定义描述了转换核心中的控制流模式在过程建模语言中的实现方式,下面介绍源语言到目标语言之间转换规则的生成策略。

首先建立源语言和转换核心之间的映射关系,遍历转换核心中的控制流模式,如果该控制流模式在源语言中的模式实现采用节点规则来描述,则说明该控制流模式在源语言中对应一种原子实现;如果该控制流模式在源语言实现中采用块规则描述,则说明该控制流模式在源语言中对应一种组合实现。按照同样的方式可以建立目标语言和转换核心之间的映射关系,然后按照下面三个策略建立源语言和目标语言之间的转换规则。

(1)当转换核心中的原子控制流模式在源语言中具有某种原子实现时,如果该模式在目标语言中也具有原子实现,则建立一条源语言到目标语言的转换规则。

(2)当转换核心中的组合模式在源语言中具有某种原子实现时,如果该模式在目标语言中也具有原子实现,则建立一条源语言到目标语言的转换规则;如果该模式在目标语言中不具有原子实现,则尝试建立源语言中的原子实现与目标语言中的组合实现的转换规则。

(3)当转换核心中的组合模式在源语言中具有某种组合实现时,如果该模式在目标语言中具有某种原子实现,则建立一条源语言到目标语言的转换规则;如果该模式在目标语言中不具有原子实现,则由于源语言,目标语言中组合实现的映射关系可以基于这些组合实现的原子模式的映射关系得到,不需要建立源语言中的组合实现到目标语言的组合实现之间的映射关系。

4　基于控制流模式的过程模型转换工具

基于本文提出的转换框架,设计了 CP-PMTT,该工具的体系结构如图 7 所示。首先将转换输入的源模型经过模型读取模块转换为可以被转换工具处理的模型内部表示,该内部表示符合本文定义的控制流结构图;然后经过控制流模式查找模块和控制流模式转换模块转换为目标模型在转换工具的内部表示;最后通过模型存储模块存储为最终的目标模型。其中,由于过程建模语言种类繁多且存储格式迥异,转换工具的模型读取、存储模块可以支持转换人员以插件的形式进行扩展。

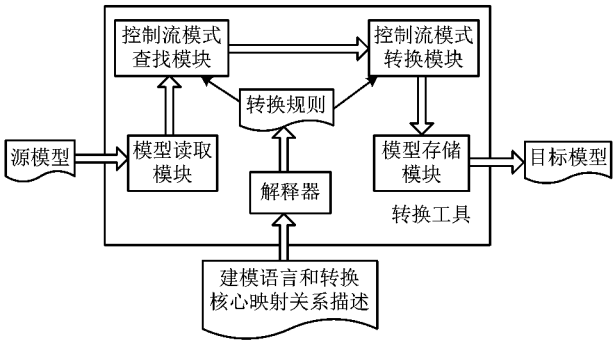


图7　过程模型转换工具的体系结构

当执行一个转换场景时，用户首先使用本文提出的控制流模式定义框架定义转换核心，为了简化用户的使用，工具已经预定义了 20 种常用的控制流模式[6]作为转换核心；然后使用 PMTL 分别描述控制流模式在源语言和目标语言中的实现形式，转换工具通过解释器生成源语言到目标语言的转换规则，这些转换规则在控制流模式查找模块和转换模块时被使用。其中在控制流模式查找模块，本文使用文献[14]提出的控制流模式查找方法，将工作流结构图规约为工作流结构树，在工作流结构树的基础上进行控制流模式查找，该查找方法的效率和正确率均相对较高。

5 案例研究

本章介绍一个 BPMN 到 YAWL 模型的转换实验，通过对 215 个 BPMN 模型的转换结果进行总结和分析，对 CP-PMTT 转换工具的实用性和正确性进行验证。

5.1 定义 BPMN，YAWL 建模语言与转换核心的映射关系

在本实验中，BPMN 到 YAWL 的转换采用转换工具中内置的 20 种常用控制流模式[8]作为转换核心，BPMN 支持 20 种控制流模式中的 18 种[15]，YAWL 则支持控制流模式中的 20 种[16]，在确定 BPMN，YAWL 支持哪些控制流模式后，用户可以使用控制流模式转换描述语言描述控制流模式在 BPMN，YAWL 中的实现形式，部分实现形式描述如表 7 所示。

表 7 BPMN，YAWL 中部分控制流模式实现描述实例

控制流模式	BPMN 实现形式	YAWL 实现形式

并行合并	pattern And_Split_Pattern:
	rule And_Split on Node
	{
	Element=And;
	InputEdgesCount=1;
	OutputEdgesCount>1;
	}
		pattern And_Split_Pattern:
		rule And_Split on Node
		{
		Element = And_Split_Task;
		InputEdgesCount=1;
		OutputEdgesCount>1;
		}

循环	pattern Loop_Pattern:
	rule Loop on Block
	{
	StartWith=Xor_Join;
	EndWith=Xor_Split;
	}
		pattern Loop_Pattern:
		rule Loop on Block
		{
		StartWith=Xor_Join;
		EndWith=Xor_Split;
		}

5.2 转换结果及分析

为了使用 BPMN 到 YAWL 的转换场景评估 CP-PMTT 的有效性，并证明 CP-PMTT 相对于基于元类的转换具有一定的优势，选取 ProM 5.0[15]中的 BPMN2YAWL 转换插件对比，该插件基于元类映射来进行 BPMN 到 YAWL 的转换。然后分别对收集到的 215 个 BPMN 过程模型进行转换，图 8 所示为这些模型的信息。

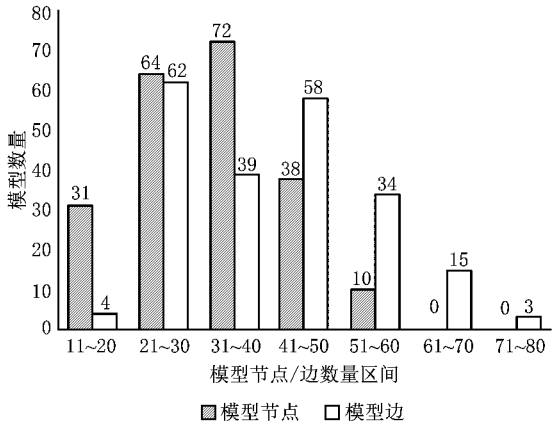


图8 BPMN模型信息

为了对比两种转换工具的转换结果，实验使用完整率和正确率两个指标来评估转换结果，这两个指标分别用来对上文的有效性和优势进行证明。首先完整率 Com 可以定量地描述为

$$Com = \frac{num_{\text{convert}}}{num_{\text{availabe}}}。$$

式中：num_{availabe} 表示实验中可用的 BPMN 模型的数量，num_{convert} 表示转换工具能够转换的模型数量。正确率 Cor 可以定量地描述为

$$Cor = \frac{num_{\text{correct}}}{num_{\text{result}}}。$$

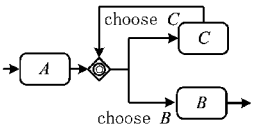
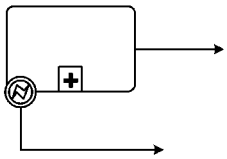
式中：num_{result} 表示在实验中转换工具能够生成的目标 YAWL 模型的数量，num_{correct} 表示生成的目标 YAWL 模型中转换结果正确的模型的数量。下面分别对两个工具的完整率和正确率进行对比说明。

5.2.1 完整率对比

在实验中，CP-PMTT 和 ProM 插件都用 ILog 建模工具(http://www-01.ibm.com/software/integration/visualization/java/)绘制的 BPMN 过程模型文件作为转换的输入，收集到的 215 个文件都可以使用 ILog 建模，因此实验中两种工具可用的 BPMN 模型文件的数量为 $num_{\text{available}}=215$。ProM

插件可以转换所有的 215 个模型,因此 ProM 插件的可转换模型数量 $num_{convert}=215$,完整率为 100%;而 CP-PMTT 不能处理 215 个模型中的 6 个,这是因为控制流模式的实现描述基于工作流结构图并进而使用环路等价算法对工作流结构图进行规约,所以无法使用工作流结构图进行表示的控制流模式无法使用 CP-PMTT 进行转换。因此 CP-PMTT 的可转换模型的数量 $num_{convert}=209$,完整率为 209/215 = 97%。表 8 所示为两类不可转换点。

表 8　CP-PMTT 的不可转换点

工作流模式	控制流模式实现	不可转换的拓扑结构	数量
WP16(def-c)		有回溯流的基于事件网关	4 个(4 个模型中)
WP20(can-c)		边界上的中间事件	4 个(2 个模型中)

5.2.2　正确率对比

在本实验中,如果生成的 YAWL 模型的控制流与 BPMN 模型的控制流一致,则称为一个正确的转换。因为实验中这种一致性的人工判定难度比较大,所以采用一种较容易判定的解决方案:如果 CP-PMTT 和 ProM 插件的转换结果一致,则假设它们都是正确的结果;对于不一致的结果,需要人工分析转换结果的差异并判断其正确性。在 209 个两种工具都能转换的模型中有 157 个一致模型和 52 个不一致模型,在 52 个不一致模型中共发现了 218 个不一致点。通过对不一致点进行分析,发现 CP-PMTT 能够正确地转换其中的 28 个模型,而 ProM 对这 52 个不一致的转换都是错误的,表 9 所示为 BP-Tool 正确转换的 28 个模型中,CP-PMTT 和 ProM 的部分不一致点。因此,CP-PMTT 的转换正确率为(157 + 28)/209 = 89%。另外对于 CP-PMTT 不能转换的 6 个模型,ProM 插件都给出了正确的转换结果,因此正确率为(6 + 157)/215 = 75%。表 10 所示为 CP-PMTT 和 ProM 插件最终结果的数据对比,从正确率可以看出,与基于元类映射的转换工具相比,使用 CP-PMTT 能够取得更加令人满意的结果。

表 9　CP-PMTT 和 ProM 转换的部分不一致点

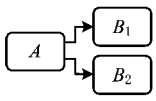
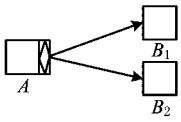
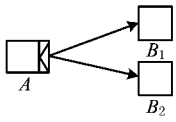
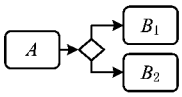
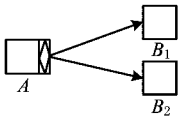
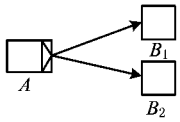
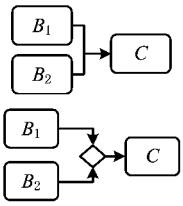
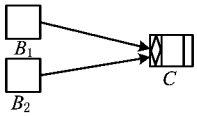
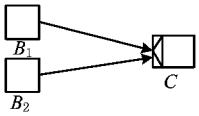
工作流模式	BPMN 控制流模式实现	ProM 插件转换结果	CP-PMTT 模型转换结果
WP2(par-spl)			
WP4(ex-ch)			
WP5(simple-m)			

表 10　CP-PMTT 和 ProM 插件转换结果对比　%

转换工具	完整率	正确率
CP-PMTT	97	89
ProM 插件	100	75

6　相关工作

目前,有很多研究学者提出多种针对特定过程模型转换场景的转换方法,例如:文献[18-21]提出

将过程模型转换到 Petri 网的方法；为了从基于 YAWL 的验证工具中获益，过程模型验证领域也有很多研究将统一建模语言(Unified Modeling Language，UML)活动图、BPMN、事件驱动过程链(Event-driven Process Chain，EPC)等过程建模语言转换到 YAWL[1-3,5,18-19,22-23]。然而，这些转换方法都主要使用基于元类的转换规则，很少包含元类与模型片段和模型片段间的映射，因此这些方法在转换某些控制流模式时有一定的困难，并且这些研究的转换方法缺少通用性，需要针对每一个转换场景定义一套独立的转换规则。

使用本文的转换工具可以将过程模型的粒度提高到控制流模式，提高过程模型转换的正确率，避免出现有些控制流模式无法得到转换结果的情况；另外，转换工具使用的转换规则通过过程建模语言和转换核心之间建立的映射关系自动生成，因此转换工具具有较强的通用性。

也有些研究者提出基于模型片段之间的映射关系定义转换规则的思想，例如：何啸等[24]提出一种改进的 QVT-R(query/view/transformation-relations)，引入模型因子支持过程模型转换中的嵌套控制流模式。扩展后的语言能够描述“模式中的活动节点由模式所引起的不确定性”，但是仍不能很好地描述由“分支节点的后继控制流具有[2. *]的多重性”引起的不确定性，因此使用改进的 QVT-R 定义过程模型转换规则仍然比较困难。文献[25]提出一种模型变形的方法，该方法定义了一个能够覆盖常见过程模型语言描述能力的集成业务过程元模型，并使用该元模型作为中间媒介实现过程建模语言之间的转换。Murzek 等提出一种基于模式的过程模型转换方法，该方法首先定义一些典型的结构模式，然后将不同语言中的不同的模型结构映射到这些共有的结构模式上。使用这种方法进行过程模型转换的转换正确率较高，但是所提出的这些结构模式比较简单，没有考虑取消、多实例等复杂模式。

本文提出的控制流模式转换框架中使用支持用户自定义控制流模式的控制流模式定义框架作为转换的核心，除了控制流模式定义框架中包括的 8 种原子控制流模式，用户也可以通过组合这些原子控制流模式定义更复杂的控制流模式，即使一种控制流模式不能由 8 种原子控制流模式加组合实现关系定义，也可以通过添加原子控制流模式，使控制流模式定义框架支持该控制流模式。控制流模式定义框架的这种可扩展性使得文本的控制流模式转换工具不但能够支持一些常用的控制流模式，而且能够支持一些过程建模语言中特有的控制流模式，从整体上提高了转换工具的转换能力。

7 结束语

为了解决基于元类映射的过程模型转换方法在转换某些控制流模式时存在转换正确性不高甚至无法得到转换结果的问题，本文提出一种基于控制流模式的过程模型转换框架，首先提出一种支持用户自定义控制流模式的控制流模式定义框架作为转换的核心，然后设计并实现了一种基于控制流模式的过程模式转换描述语言，用以描述源语言、目标语言中控制流模式的实现形式，进而建立源语言、目标语言和转换核心的映射关系，自动生成源语言到目标语言的转换规则。基于上述框架，实现了一种基于控制流模式的过程模型转换工具。为了验证转换工具的有效性，使用工具进行 BPMN 到 YAWL 的转换，根据转换结果证明了使用基于控制流模式的过程模型转换工具是有效的，并且与基于元类映射的转换工具相比具有较高的正确率。

因为本文选取工作流结构图表示过程模型中的控制流结构，所以难以支持一些特定语言中的控制流模式(如表 8)。此外，过程模型转换描述语言对控制流模式实现形式的描述也有一定的局限性，例如无法描述 BPMN 中使用“过程”包含两个不相关“任务”的形式表示的并行分支模式。为了提高基于控制流模式转换方法的可适用性，并提高对一些复杂控制流模式实现形式的描述能力，后续工作将集中在对工作流结构图的改进和控制流模式转换描述语言的扩展上。

参考文献：

[1] SAC. GB/T 19001-2008 Quality management systems-requirements[S]. Beijing: Standard Press of China, 2009(in Chinese). [中国国家标准管理委员会. GB/T 19001-2008 质量管理体系要求[S]. 北京：中国标准出版社，2009.]

[2] YE Jianhong, SUN Shixin, WEN Lijie, et al. Transformation of BPMN to YAWL[C]//Proceedings of the International Conference on Computer Science and Software Engineering. Washington, D. C., USA: IEEE, 2008.

[3] WEIDICH M, WESKE M. Business process modeling mota-tion[M]. Berlin, Germany: Springer-Verlag, 2010.

[4] VAN DER AALST W M P, TER HOFSTEDE A H M. YAWL: yet another workflow language[J]. Information Sys-

tems,2005,30(4): 245-275.

[5] DECKER G, DIJKMAN R, DUMAS M, et al. Transforming BPMN diagrams into YAWL nets[M]. Berlin, Germany: Springer-Verlag, 2008: 386-389.

[6] VAN DER AALST W M P, TER HOFSTEDE A H M, KIEPUSZEWSKI B, et al. Workflow patterns[J]. Distributed and Parallel Databases, 2003, 14(1): 5-51.

[7] CUMBERLIDGE M. Business process management with JBoss jBPM[M]. Birmingham, UK: Packt Publishing Ltd, 2007.

[8] ZHOU Bosheng, ZHANG Sheying. Visual process modeling language VPML[J]. Journal of Software, 1997, 8(S): 535-545.

[9] MURZEK M, KRAMLER G, MICHLMAYR E. Structural patterns for the transformation of business process models [C]//Proceedings of the 10th IEEE International Enterprise Distributed Object Computing Conference Workshops. Washington, D. C. USA: IEEE, 2006: 18.

[10] VAN DER AALST W M P, BARROS A P, TER HOFSTEDE A H M, et al. Advanced workflow patterns[C]//Cooperative Information Systems. Berlin, Germany: Springer-Verlag, 2000: 18-29.

[11] WEN Lijie, WANG Jianmin, SUN Jiaguang. Modeling workflow patterns using coloured Petri nets[J]. Computer Science, 2006, 33 (6): 135-139(in Chinese). [闻立杰, 王建民, 孙家广. 用着色 Petri 网建模工作流模式[J]. 计算机科学, 2006, 33(6): 135-139.]

[12] SADIQ W, ORLOWSKA M E. Analyzing process models using graph reduction techniques[J]. Information Systems, 2000, 25(2): 117-134.

[13] JOHNSON R, PEARSON D, PINGALI K. The program structure tree: computing control regions in linear time[C]// Proceedings of SIGPLAN Conference on Programming Language Design and Implementation ACM SigPlan Notices. New York, N. Y., USA: ACM, 1994, 29(6): 171-185.

[14] HAN Z, GONG P, ZHANG L, et al. Definition and detection of control-flow anti-patterns in process models[C]//Proceedings of the 2013 IEEE 37th Annual Computer Software and Applications Conference. Washington, D. C. USA: IEEE, 2013: 433-438.

[15] WOHED P, VAN DER AALST W M P, DUMAS M, et al. On the suitability of BPMN for business process modelling [M]. Berlin, Germany: Springer-Verlag, 2006.

[16] VAN DER AALST W M P. Verification of workflow nets [C]//Proceedings of Application and Theory of Petri Nets 1997. Berlin, Germany: Springer-Verlag, 1997: 407-426.

[17] Process Mining Group. ProM Tools[EB/OL]. [2014-10-16]. http://promtools.org/prom5.

[18] 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.

[19] LÓPEZ-GRAO J P, MERSEGUER J, CAMPOS J. From UML activity diagrams to Stochastic Petri nets: application to software performance engineering[C]//Proceedings of ACM SIGSOFT Software Engineering Notes. New York, N. Y., USA: ACM, 2004, 29(1): 25-36.

[20] ZHA Haiping, VAN DER AALST W M P, WANG Jianmin, et al. Verifying workflow processes: a transformation-based approach[J]. Software and System Modeling, 2011, 10(2): 253-264.

[21] ZHA H, YANG Y, WANG J, et al. Transforming XPDL to Petri nets[C]//Proceedings of Business Process Management Workshops. Berlin, Germany: Springer-Verlag, 2008: 197-207.

[22] MENDLING J, MOSER M, NEUMANN G. Transformation of yEPC business process models to YAWL[C]//Proceedings of the 2006 ACM Symposium on Applied Computing. New York, N. Y., USA: ACM, 2006: 1262-1266.

[23] BROGI A, POPESCU R. From BPEL processes to YAWL workflows[M]. Web Services and Formal Methods. Berlin, Germany: Springer-Verlag, 2006: 107-122.

[24] HE Xiao, MA Zhiyi, ZHANG Yan, et al. Extending QVT relations for business process model transformation[J]. Journal of Software, 2011, 22(2): 195-210(in Chinese). [何　啸, 麻志毅, 张　岩, 等. 扩展 VT Relations 实现业务流程模型的转换[J]. 软件学报, 2011, 22(2): 195-210.]

[25] MURZEK M, KRAMLER G. The model morphing approach-horizontal transformations between business process models[C]//Proceedings of the 6th International Conference on Perspecitves in Business Information Research-BIR 2007. 2007, 2007: 88-103.

作者简介:

邵真禄(1991—),男,河南确山人,硕士研究生,研究方向:过程模型转换,E-mail:shaozhenlu@126.com;

+张　莉(1968—),女,四川成都人,教授,博士生导师,研究方向:软件体系结构、企业过程、过程工程、需求工程,通信作者,E-mail:lily@buaa.edu.cn;

凌济民(1989—),男,江西南昌人,博士研究生,研究方向:过程建模。