软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2018,29(11):3260−3277 [doi: 10.13328/j.cnki.jos.005476]

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

基于行为特征的语义工作流修正算法*

孙晋永[1,2], 古天龙[2], 闻立杰[3], 钱俊彦[2], 刘华东[2]

[1](西安电子科技大学 计算机科学与技术学院,陕西 西安 710071)

[2](广西可信软件重点实验室(桂林电子科技大学),广西 桂林 541004)

[3](清华大学 软件学院,北京 100084)

通讯作者: 古天龙, E-mail: cctlgu@guet.edu.cn

摘 要: 工作流修正是工作流重用的重要任务.目前,在基于工作流的可重用片段——stream 的语义工作流修正中,当工作流 stream 库中不存在与检索语义工作流中的工作流 stream 结构相似的 stream 时,无法修正检索语义工作流.针对这种情况,提出了一种改进方法——基于 stream 行为特征的语义工作流修正算法.使用任务紧邻关系集表达 stream 的行为特征.对于检索语义工作流的每个 stream(称为查询 stream),使用锚集合数据索引和 stream 匹配规则过滤工作流 stream 库得到候选匹配 stream 集;之后,基于变更请求和 stream 的行为相似性对候选 stream 集进行验证,得到需替换的查询 stream 和最符合变更请求并与它足够行为相似的匹配 stream;然后,使用每个匹配 stream 替换对应需替换的查询 stream 以逐步修正检索语义工作流中的缺陷;最后得到修正语义工作流.实验结果表明,与现有的基于工作流 stream 的修正算法相比,该算法得到了整体质量更好的修正语义工作流集,其适应性更好.该修正算法能够为业务过程管理人员进行适应新业务需求的工作流变更提供较好质量的参考语义工作流,对提高业务过程管理中工作流重用的效率和质量有较大的帮助.

关键词: 工作流重用;语义工作流;修正;工作流 stream;行为特征

中图法分类号: TP311

中文引用格式: 孙晋永,古天龙,闻立杰,钱俊彦,刘华东.基于行为特征的语义工作流修正算法.软件学报,2018,29(11):3260−3277. http://www.jos.org.cn/1000-9825/5476.htm

英文引用格式: Sun JY, Gu TL, Wen LJ, Qian JY, Liu HD. Adaptation algorithm of semantic workflows based on behavioral characteristics. Ruan Jian Xue Bao/Journal of Software, 2018,29(11):3260−3277 (in Chinese). http://www.jos.org.cn/1000-9825/5476.htm

Adaptation Algorithm of Semantic Workflows Based on Behavioral Characteristics

SUN Jin-Yong[1,2], GU Tian-Long[2], WEN Li-Jie[3], QIAN Jun-Yan[2], LIU Hua-Dong[2]

[1](School of Computer Science and Technology, Xidian University, Xi'an 710071, China)

[2](Guangxi Key Laboratory of Trusted Software (Guilin University of Electronic Technology), Guilin 541004, China)

[3](School of Software, Tsinghua University, Beijing 100084, China)

* 基金项目: 国家自然科学基金(61572146, U1501252, 61562015, 61862016); 广西自然科学基金(2016GXNSFDA380006, 2017GXNSFAA198283); 广西高等学校高水平创新团队及卓越学者计划; 广西可信软件重点实验室基金(KX201723)

Foundation item: National Natural Science Foundation of China (61572146, U1501252, 61562015,61862016); Guangxi Natural Science Foundation (2016GXNSFDA380006, 2017GXNSFAA198283); High Level of Innovation Team of Colleges and Universities in Guangxi and Outstanding Scholars Program; Guangxi Key Laboratory of Trusted Software (KX201723)

本文由面向智能制造的业务过程管理与服务技术专题特约编辑王建民教授、刘建勋教授推荐.

收稿时间: 2017-07-19; 修改时间: 2017-09-16; 采用时间: 2017-11-14; jos 在线出版时间: 2017-12-06

CNKI 网络优先出版: 2017-12-06 15:37:19, http://kns.cnki.net/kcms/detail/11.2560.TP.20171206.1536.023.html

Abstract: Workflow adaptation is an important task of workflow reuse. During semantic workflow adaptation based on workflow streams, i.e., the reusable segments of semantic workflows, the absence of workflow streams structurally similar to the streams of the retrieved semantic workflow in the workflow streams repository leads to unachievable workflow adaptation. Focusing on the problem, this paper proposes an improved method, i.e., an adaptation algorithm of semantic workflows based on behavioral characteristics of workflow streams. The set of task adjacency relations is used to express the workflow streams' behavioral characteristics. First, for each stream of the retrieved semantic workflow (called query stream), the data index of the anchor set and stream matching rules are used to filter the workflow stream repository to obtain the matching stream candidates.Then, these stream candidates are verified with the change request and the behavioral similarity metric, to obtain the query streams that need to be substituted and the corresponding matching streams that are most coincident with the change request and behaviorally similar to them. Next, each matching stream is used to substitute the query stream in the retrieved workflow to gradually adapt defects of retrieved workflow. Finally, the adapted semantic workflow is obtained. The experimental results show that the proposed adaptation algorithm achieves the adapted semantic workflow set with higher overall quality and has better adaptability compared with existing adaptation algorithm based on workflow streams. The adaptation algorithm can provide semantic workflows of higher quality for business processes managers for references when adapting workflows to meet new business requirements, and is helpful for the improvement of the efficiency and quality of workflow reuse in business process management (BPM).

Key words: workflow reuse; semantic workflow; adaptation; workflow stream; behavioural characteristics

目前,工作流的应用已经从最初的领域——业务过程扩展到新的领域,如电子商务、医疗、软件开发、科学分析、信息集成甚至烹饪等.对柔性工作流的需求不断增加,促使研究者寻求新方法来支持领域专家创建、监控和修正工作流[1].近年来,研究者提出基于知识的业务过程管理,将基于事例推理(case-based reasoning,简称CBR)引入业务过程管理中,提出了面向过程 CBR(process-oriented CBR,简称 POCBR).POCBR 使用工作流案例记录过去的工作流建模、执行活动的经验知识,进行推理以支持领域专家创建、修正和重用工作流[1].

工作流是业务过程的自动化实现,基于知识的工作流管理需要以领域知识为背景.语义工作流[1]作为一种基于知识的工作流,提供了充足的语义、数据或资源信息,为高效率的工作流重用提供了便利.工作流重用通过复用工作流库中的相似工作流或其片段来构建新工作流,它的两个重要任务是检索相似工作流和修正检索工作流[2].当检索到的最相似工作流不能满足查询中的特殊需求时,可以对它进行修正.目前,工作流修正基本上有两种方法:手工修正和自动修正.手工修正方法对工作流建模人员的经验要求较高,而自动修正方法需要充分利用工作流库的知识.当前,语义工作流重用是工作流重用研究的热点之一.目前,对相似语义工作流检索的研究已经取得了较多成果,但对语义工作流修正方法的研究还较少.本文关注语义工作流的自动修正方法.

目前,用于 CBR 案例修正的变换修正法[3]已被引入工作流修正中.Minor 等人[4,5]提出了基于修正案例的语义工作流变换修正法.该方法基于预先建立的修正知识(即修正案例)实现,但获取工作流的修正案例是一个难题.Müller 等人[6]提出了基于工作流泛化的语义工作流变换修正法,在引入领域本体知识前提下,通过泛化工作流的部分元素的语义描述得到已泛化工作流,通过特化已泛化的工作流来完成语义工作流修正.该方法需要预先泛化语义工作流库,计算量较大.Müller 等人[2]提出了一种无需预先建立的修正知识、基于工作流 stream 的语义工作流复合修正方法.该方法将语义工作流库中的每个工作流分解成有意义的片段——工作流 stream,作为可重用的组件,组成工作流 stream 库.检索语义工作流中的工作流 stream 被来自工作流 stream 库、最符合变更请求且足够结构相似的 stream 替换,最终得到修正语义工作流.Müller 等人[7]提出了基于工作流 streamlet 修正算子(包括插入、删除和交换算子)的语义工作流修正方法.工作流 streamlet 是由处理某个数据对象的任务节点及其数据对象组成的一个工作流片段.使用修正算子链可以将检索语义工作流转换成修正语义工作流.该方法需要预先学习工作流 streamlet 修正算子集合,该过程的计算量较大.以上两种方法可以得到与工作流库中的语义工作流的质量相当的修正语义工作流,但其适用前提是在工作流库中存在符合变更请求并且足够结构相似的 stream 或 streamlet.如果不存在这样的 stream 或 streamlet,则不能完成语义工作流修正.

研究者通常使用过程模型的有向标签图的拓扑结构或任务节点的连接关系来刻画过程模型结构特征.相比于结构特征,过程模型的动态执行行为刻画了其本质特性[8].试图通过获取过程模型所有的执行轨迹来准确地描述该业务过程模型的执行行为是行不通的.研究者一般使用行为特征来近似刻画过程模型的执行行为.目

前,已经提出了多种刻画过程模型行为特征的方法,如变迁紧邻关系集[9]、依赖图[10]、主变迁序列集[11]、因果足迹[12]和行为轮廓[13]、有代表性的轨迹集[14]、任务发生关系集[15]等.与其他方法相比,变迁紧邻关系(transition adjacency relations,简称 TAR)集具有形式简洁、计算效率高和准确性较高等优点;并且相比整个工作流,工作流片段的 TAR 集更易于获取.

针对基于工作流 stream 的语义工作流修正方法[2]的不足,本文引入任务紧邻关系(task adjacency relations,简称 TAR)集来刻画 stream 的行为特征;提出了一种基于 stream 行为特征的语义工作流修正方法.结合领域数据知识构建工作流 stream 库的锚集合数据索引 ASDataIndex,提出了 stream 匹配规则和 stream 的行为相似性方法.在此基础上,设计了从工作流 stream 库中检索查询 stream 的匹配 stream 的算法。对于检索语义工作流中的每个 stream,先使用 ASDataIndex 和 stream 匹配规则过滤工作流 stream 库得到候选匹配 stream 集;然后,基于变更请求和 stream 行为相似性对候选 stream 集进行验证,得到需替换的 stream 和最符合变更请求并与它足够行为相似的匹配 stream;接着使用匹配 stream 替换需替换的 stream 以逐步修正检索语义工作流的缺陷;最后得到修正语义工作流.与基于 stream 的修正方法[2]相比,本文的基于行为特征的修正方法提高了基于 stream 修正方法的适用性和健壮性,提高了修正语义工作流的质量,为语义工作流重用的效率提供了更可靠的保障.

1 基本概念

为传统工作流的任务结点增加语义描述和输入/输出数据对象,为边增加语义描述,使工作流同时包含控制流、数据流和语义约束,可以得到语义工作流[1].

1.1 语义工作流

定义 1(语义工作流[1]). 语义工作流可以形式化为语义标注有向图 $SW=(N,E,S,\tau)$.其中,$N=N_T\cup N_C\cup N_D$,N_T 是任务节点的集合,N_C 是控制流节点的集合,N_D 是数据节点集合.$E\subseteq N\times N$ 是边的集合;$S:N\cup E\to\Sigma$将节点或边映射为语义描述;Σ是一个领域相关的语义元数据语言.$\tau:N\cup E\to type$ 将每个节点或边映射为来自集合Ω的一个类型.集合$\Omega=\{task\ node,data\ node,control\text{-}flow\ node,control\text{-}flow\ edge,dataflow\ edge\}$.并记 $SN=N_T\cup N_C$ 为顺序节点的集合,$CE\subseteq SN\times SN$ 为控制流边的集合,$DE\subseteq(N_D\times SN)\cup(N_D\times SN)$为数据流边的集合.

为了表述简便,本文仅以面向控制流的(control-flow oriented)业务工作流为研究对象;将其形式化表示为语义工作流,研究业务工作流的修正方法.

例 1:图 1 是描述意大利面食 Baked spaghetti 烹饪过程的语义工作流 SW_1[16].

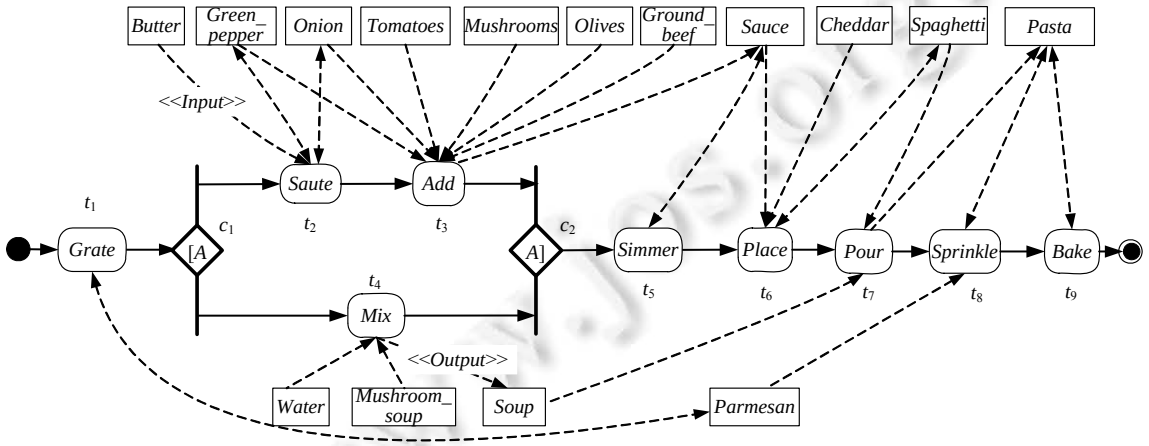


Fig.1 Semantic workflow SW_1

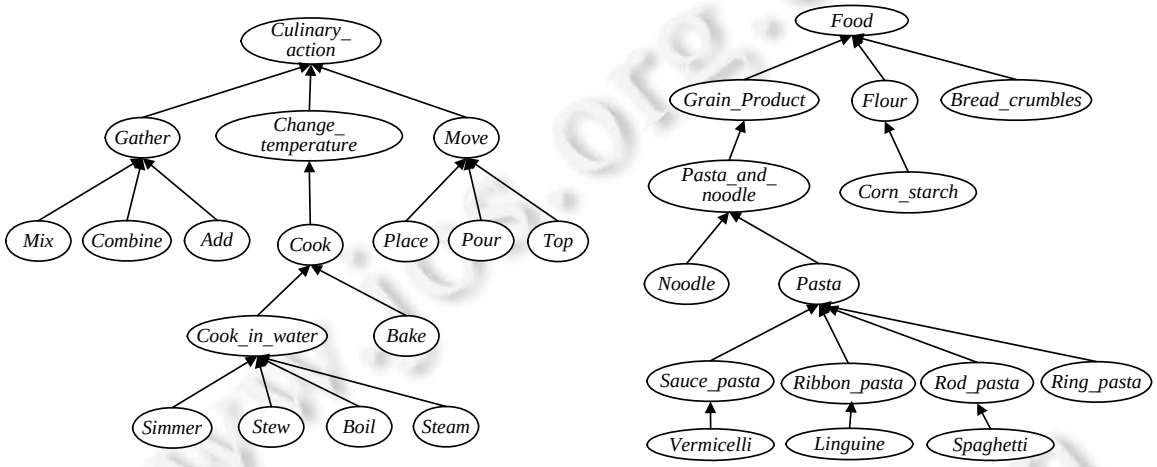
图 1 语义工作流 SW_1

在图 1 中,带箭头的实线表示控制流边,为了简洁,省略了它们的语义描述 *Control-flow*;带箭头的虚线表示数据流边,*Input* 为输入数据流边的语义描述,*Output* 为输出数据流边的语义描述.在图 1 中,分别只标注了 1 处输入数据流边和输出数据流边的语义描述,省略了其余数据流边的标注.

语义工作流的任务结点以语义描述指示的方式消耗了输入数据对象,生成了输出数据对象.在引入领域本体表示领域知识的前提下,任务节点和数据对象节点的语义描述分别为领域任务本体和数据本体中的语义概念.在本文的烹饪工作流实例中,以任务本体表示烹饪领域的烹饪动作知识,以数据本体表示食材知识.故在烹饪工作流的语义标注有向图中,数据对象节点代表了其语义描述所指示的食材,而任务节点则代表了对这些食材采用其语义描述所指示的烹饪动作.

控制流边集 *CE* 包含了顺序节点间的一个严格偏序关系.对于顺序节点 $s_1,s_2\in SN$,如果 s_1 在 s_2 之前执行,则记 $s_1\prec s_2$.如图 1 中,有 $t_1\prec c_1,t_2\prec t_3,t_3\prec c_2$.对 $s\in SN$,记 $s\in[s_1,s_2]$当且仅当 $s_1\prec s\wedge s\prec s_2$[2].

例 2:图 2 是烹饪领域的任务本体和数据本体的片段.



(a) 任务本体 (b) 数据本体

Fig.2 Task ontology and data ontology

图 2 任务本体和数据本体

在图 2(a)中,*Cook* 是 *Bake*,*Simmer* 的泛化概念(父概念),*Bake*,*Simmer* 是 *Cook* 的特化概念(子概念),记为 $Bake\sqsubseteq Cook$,$Simmer\sqsubseteq Cook$.特别地,称 *Mix*,*Combine*,*Add* 为 *Gather* 的一步特化概念,记为 $Mix\sqsubseteq^{*}Gather$,$Combine\sqsubseteq^{*}Gather$ 和 $Add\sqsubseteq^{*}Gather$;相反地,称 *Gather* 为 *Mix*,*Combine*,*Add* 的一步泛化概念.假设 C_1 和 C_2 是图 2 所示本体的语义概念,则 C_1 和 C_2 的相似性为

$$sim(C_1,C_2)=\begin{cases}1, & \text{if } C_1\sqsubseteq^{*}C_2\\ \dfrac{2\cdot depth(LCS(C_1,C_2))}{depth(C_1)+depth(C_2)}, & \text{otherwise}\end{cases}\tag{1}$$

其中,$LCS(C_1,C_2)$为 C_1,C_2 的最具体公共父概念,$depth(C)$为语义概念 C 在本体中的深度.如图 2(b)中,*Food* 的深度为 1,*Pasta* 的深度为 4.

如果语义工作流的每个并发、选择或循环控制流块均具有一个明确的开节点和一个闭节点,并且这些控制流块可以嵌套但不能交错,则该语义工作流被称为块结构化[17]的(block-structured)语义工作流.为了限定研究范围,本文假定所研究的语义工作流是块结构化的.

1.2 语义工作流stream

定义 2(部分语义工作流[2]**).** 语义工作流 $SW=(N,E,S,\tau)$的一个部分语义工作流是一个块结构化的语义工作

流 $SW'=(N',E'\cup CE'_+,S,\tau)$,其中, $N'=N'_T\cup N'_C\cup N'_D\subseteq N, N'_D\subseteq N_D$ 是 SW'的数据节点,$E'\subseteq E\cap(N'\times N')$是连接 N'中两个节点的边集合, CE'_+ 是为了保持顺序节点的执行关系而加入的附加边, $CE'_+=\{(n_1,n_2)\in SN'\times SN'|n_1\prec n_2\wedge\neg\exists n\in SN:(n_1,n)\in CE'\vee(n,n_2)\in CE'\vee n\in[n_1,n_2]\}$,$S,\tau$同定义 1.

通常,控制流节点也是部分语义工作流的一个部分.如图 3 所示的部分语义工作流 PW 中的双线箭头,就是为了保持顺序节点的执行关系而加入的附加边.

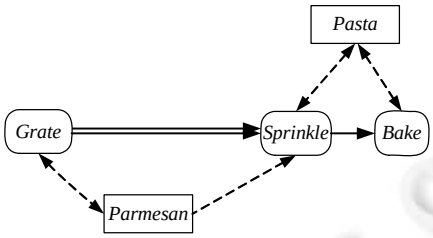


Fig.3 Partial semantic workflow PW

图 3 部分语义工作流 PW

定义 3(生产者任务节点[2]). 在语义工作流中,生成某个数据对象但不消耗该数据对象的任务结点被称为生产者(creator)任务节点.生产者任务节点的集合可以形式化为 $CT=\{t\in N_T|\exists d\in N_D:((t,d)\in DE\wedge(d,t)\notin DE)\}$.

例 3:语义工作流 SW_1 中的工作流 stream,如图 4 所示.

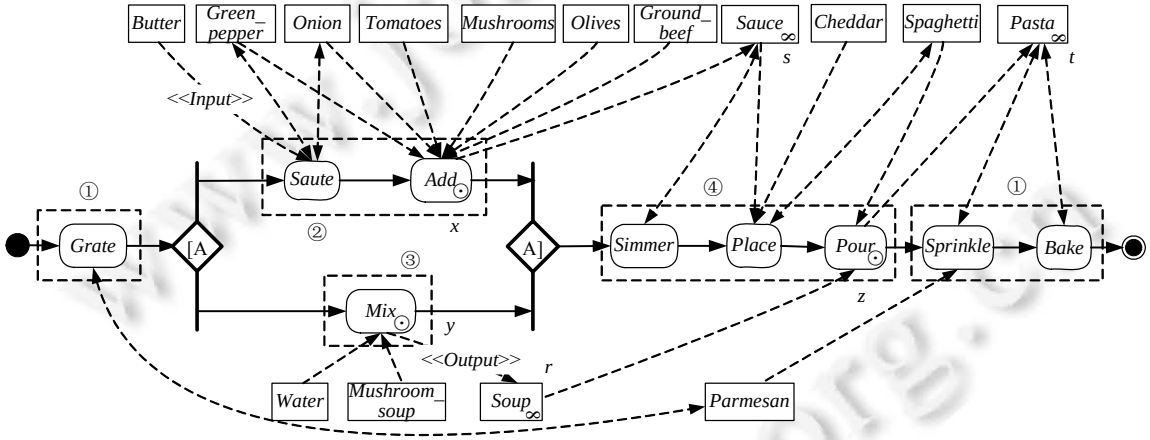


Fig.4 Streams of semantic workflow SW_1

图 4 语义工作流 SW_1 的工作流 stream

图 4 中的被标记⊙的任务节点 x,y,z 均为生产者任务节点.本质上,生产者任务节点完成了整个工作流的部分目标:它消耗多个数据对象,生成新的数据对象.对于 $t_1,t_2\in N_T$,如果 $t_1\prec t_2$ 并且 t_2 消耗了 t_1 生成的一个数据节点 d,则称 t_1,t_2 是数据流连接的[2],记为 $t_1\overset{d}{\mapsto}t_2$.如果 t_1,t_2 传递数据流连接的,则记 $t_1\overset{*}{\mapsto}t_2$.

定义 4(工作流 stream[2]). 语义工作流 SW 的一个划分为 $S^W=\{S_1^W,S_2^W,...,S_i^W,S_{i+1}^W,...,S_n^W\}$,其中, S_i^W 为一个部分语义工作流,$i\in\{1,2,...,n-1\}$.如果对于$\forall t\in N_T$ 都被包含在一个且仅一个 S_i^W 中,使得$\forall t\in N_T$ 均满足传递的数据流连接,并且每个生产者任务节点$\forall t\in CT$ 都是 S_i^W 的终节点,则该部分工作流 S_i^W 被称为一个工作流 stream.

由定义 4 中,每个生产者任务节点都是一个工作流 stream 的终节点,这只是充分条件.也存在终节点不是生产者任务节点的工作流 stream.如图 4 中,标号①,②,③,④的部分语义工作流均为工作流 stream,分别记为 S_1,S_2,S_3,S_4,其中,S_1 是终节点不是生产者任务节点的工作流 stream.在不影响准确性的前提下,为了简便,下文在一些描

述中将工作流 stream 简称为 stream.

在使用 Müller 等人[2]的方法进行语义工作流修正前,首先需要从语义工作流库中获取 stream 集合.对一个语义工作流,得到它的 stream 集合的方法[2]如下:对于$\forall y\in CT$,首先识别出一个满足数据传递依赖的任务节点集$T_y=\{t\in N_T|t\overset{*}{\mapsto}y\wedge t\notin CT\wedge\neg\exists x\in CT:(t\overset{*}{\mapsto}x)\wedge\neg(y\prec x)\}\cup\{y\}$.$y$,$T_y$ 及其所连接的控制流和数据流边组成了一个 stream.然后连接剩余节点,组成一个工作流 steam.对语义工作流库中的每个语义工作流进行分解,保存得到的 stream,组成了工作流 stream 库.

定义 5(锚集合[2]). 设 St 为语义工作流 SW 的 stream,St 的锚集合是在它的输入/输出数据对象集合中,被 SW 中不同于 St 的 stream 的任务节点生产或消耗的数据对象节点集合 AS.表示为 $AS=AS_{in}\cup AS_{out}$,其中,$AS_{in}=\{d\in D_S|(\exists t\in N_T\setminus T_S\wedge\exists t_S\in T_S):t\overset{d}{\mapsto}t_S\}$,$AS_{out}=\{d\in D_S|(\exists t\in T\setminus T_S\wedge\exists t_S\in T_S):(t_S\overset{d}{\mapsto}t)\}$.其中,$AS_{in}$ 为输入数据流对应的锚集合,简称输入锚集合;AS_{out} 为输出数据流对应的锚集合,简称输出锚集合.D_S,T_S 分别为 St 的数据对象节点集合和任务节点集合.

图 3 中的被标记"∞"的数据对象节点 r,s,t 均为锚集合中的数据对象,其中,$\{r\}$为 S_3 的输出锚集合,$\{r,s\}$为 S_4 的输入锚集合,其他情况类似.两个 stream 是匹配的或可替换的(substitutable),仅当它们的锚集合是匹配的.但这个替换过程还必须考虑锚集合内数据对象的处理方式,这也是要求匹配 stream 必须结构或行为相似的原因.

2 基于工作流 stream 行为特征的语义工作流修正算法

针对 Müller 等人[2]的方法的不足,本文提出了基于 stream 行为特征的语义工作流修正算法.算法由两阶段构成:检索符合变更请求且行为相似的匹配 stream 阶段和修正检索语义工作流阶段.思路如下:当检索到的最相似工作流不能满足查询中的特殊需求时,用户可以针对特殊需求对它进行修正.这些特殊需求被转换成对检索工作流的变更请求(change request).不满足这些特殊需求即与变更要求不一致.设 SW 为检索到的最相似且不满足特殊需求的语义工作流,为了对 SW 进行修正,首先把 SW 分解成 stream 集合 WS_0.然后,对于 WS_0 中的每个 stream St_q,从工作流 stream 库中过滤出 St_q 的匹配 stream 集 RS.接着,基于变更请求和 stream 行为相似性确定 St_q 是否需要被替换;如果需要,则从 RS 中检索出最符合变更请求并与 St_q 足够行为相似的匹配 stream St_m.最后,使用每个 St_m 替换 SW 中的对应 St_q 以修正检索工作流的缺陷.其中,检索最符合变更请求且与 St_q 足够行为相似的匹配 stream 是本文的修正算法的重点,也是核心工作.

2.1 检索与变更请求一致且行为相似的匹配工作流stream

为了便于阐述匹配 stream 检索算法,先介绍 stream 匹配的相关概念.本节中的查询 stream 即为待修正的检索语义工作流中的 stream.

2.1.1 工作流 stream 匹配规则

在检索与查询 stream 匹配的 stream 时,有时不能得到锚集合完全一致的 stream.这会使检索语义工作流无法修正.这时可以考虑锚集合相近的 stream.因而本文定义了泛化 stream 和 stream 匹配的概念.

定义 6(泛化工作流 stream). 工作流 stream $St_1=(N_1,E_1,S_1,\tau)$是工作流 stream $St_2=(N_2,E_2,S_2,\tau)$的泛化工作流 stream,当且仅当存在 St_1,St_2 间的图同构 $I:N_2\to N_1$,使得对于$\forall n_2\in N_2,S_2(n_2)\sqsubseteq^*S_1(I(n_2))$;同时,称 St_2 是 St_1 的特化,记作 $St_2\sqsubseteq^*St_1$.

工作流 stream 对应的有向图规模较小,本文对 Ullmann 算法[18]进行改造来检验 stream 间的图同构关系.参考第 1.1 节的本体中语义概念间的泛化/特化关系,本文规定:如果 $St_2\sqsubseteq^*St_1$,则在语义工作流修正中可以使用 St_2 替换 St_1.例如,假设在领域任务本体中有 $Sprinkle\sqsubseteq^*Garnish$,$Bake\sqsubseteq^*Cook$;在领域数据本体中有 $Parmesan\sqsubseteq^*Cheese$.如图 5 为 stream S_5,S_1 为图 4 中 SW_1 的 stream,则由定义 6 可得,$S_1\sqsubseteq^*S_5$.

定义 7(语义描述集合的泛化). 设集合 SD_1,SD_2 为数据对象节点的语义描述的集合,如果 SD_1,SD_2 间存在一一对应的映射 f,并且对$\forall a\in SD_1,\exists b=f(a)\in SD_2$ 满足 $b\sqsubseteq^*a$,则称 SD_1 是 SD_2 的一步泛化,SD_2 是 SD_1 的一步特化,记为 $SD_2\sqsubseteq^*SD_1$.

例如,假设领域数据本体中有 $Beef\sqsubseteq^{*}Meat$,$Onion\sqsubseteq^{*}Vegetable$,若有数据对象语义描述的集合 B_1={*Beef*, *Onion*},B_2={*Meat*,*Vegetable*},则 B_2 是 B_1 的一步泛化,$B_1\sqsubseteq^{*}B_2$.

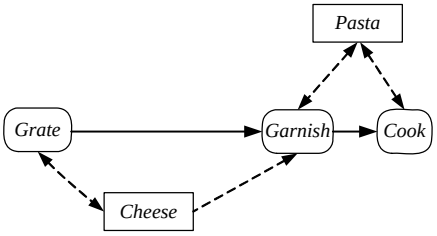


Fig.5 Workflow stream S_5

图 5 工作流 stream S_5

基于定义 7,定义基于锚集合匹配的工作流 stream 匹配概念.

定义 8(工作流 stream 匹配). 设查询工作流 stream St_q 的锚集合为 AS_1,某工作流 stream St_2 的锚集合为 AS_2. A_1,A_2 分别为 St_q,St_2 的输入锚集合对应的语义描述集合,B_1,B_2 分别为 St_q,St_2 的输出锚集合对应的语义描述集合:

1) 精确匹配:如果 $A_1=A_2$,并且 $B_1=B_2$,即 A_1 与 A_2,B_1 与 B_2 完全等价,则称 St_2 与 St_q 精确匹配;
2) 泛化匹配:如果 $St_2\sqsubseteq^{*}St_q$,即 St_2 比 St_q 更具体,则称 St_2 与 St_q 泛化匹配;
3) 嵌入匹配:如果 $A_2\sqsubseteq^{*}A_1$,并且 $B_2\sqsubseteq^{*}B_1$,即 A_2 比 A_1,B_2 比 B_1 更具体,则称 St_2 与 St_q 嵌入匹配;
4) 完全匹配:如果 $A_1\sqsubseteq^{*}A_2$,并且 $B_1\sqsubseteq^{*}B_2$,即 A_1 比 A_2,B_1 比 B_2 更具体,则称 St_2 与 St_q 完全匹配;
5) 潜在匹配:如果 $A_2\sqsubseteq^{*}A_1\vee A_1\sqsubseteq^{*}A_2,B_2\sqsubseteq^{*}B_1\vee B_1\sqsubseteq^{*}B_2$ 均不成立,但$\exists a_1\in A_1\wedge\exists a_2\in A_2\wedge a_1\sqsubseteq^{*}a_2$ 或者$\exists b_1\in B_1\wedge\exists b_2\in B_2\wedge b_1\sqsubseteq^{*}b_2$ 成立,则称 St_2 与 St_q 潜在匹配;
6) 不匹配:如果 A_1,A_2,B_1,B_2 不满足以上 5 种情况,则称 St_2 与 St_q 不匹配.

假如某查询 stream 的要求是输入锚集合包含语义描述为 *Meat* 的数据对象节点,由于 $Pork\sqsubseteq^{*}Meat$,$Beef\sqsubseteq^{*}Meat$,故输入锚集合包含语义描述为 *Pork* 或 *Beef* 的数据对象节点的 stream 都应该准确地满足该要求.这是嵌入匹配类型.又如,假设在领域任务本体中有 $Stir\sqsubseteq^{*}Mix$,在领域数据本体中有 $Soft\_water\sqsubseteq^{*}Water$,$Cream\_mushroom\_soup\sqsubseteq^{*}Mushroom\_soup$,$Mushroom\_soup\sqsubseteq^{*}Soup$.如图 6 所示为 stream S_6,S_3 为图 4 中 SW_1 的 stream,则由定义 8 可得 S_6,S_3 是嵌入匹配.又由于 $S_6\sqsubseteq^{*}S_3$,同时 S_6,S_3 也是泛化匹配.

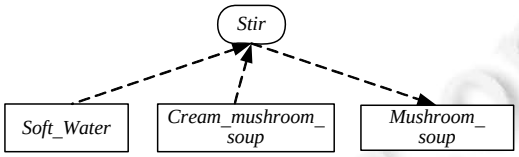


Fig.6 Workflow streams S_6

图 6 工作流 stream S_6

基于定义 8,本文规定工作流 stream 匹配规则如下:先选择精确匹配的 stream,然后选择泛化匹配的 stream,再选择嵌入匹配的 stream,最后选择完全匹配的 stream.但由于完全匹配、潜在匹配和不匹配类型不能给出准确满足要求的 stream,故本文只研究精确匹配、泛化匹配和嵌入匹配这 3 种类型.

2.1.2 基于锚集合数据索引和匹配规则获取候选匹配工作流 stream

建立索引是提高检索效率的有效方法,本节构建 stream 库的锚集合数据索引 ASDataIndex.使用 ASDataIndex 并结合 stream 匹配规则,可以从 stream 库中检索到与查询 stream 匹配的候选匹配 stream 集合.该过程被称为过滤[19]匹配 stream.

定义 9(锚集合数据索引). 工作流 stream 库的锚集合数据索引 ASDataIndex 是一个倒排表.它的每个节点

为形如($data$,$data.list_{in}$,$data.list_{out}$)的项,建立了从 $data$ 到工作流 stream 集合的映射.其中,$data$ 为锚集合的数据对象,$data.list_{in}$ 为输入锚集合中包含 $data$ 的工作流 stream 集合.$data.list_{out}$ 为输出锚集合中包含 $data$ 的工作流 stream 集合.

索引 ASDataIndex 的结构如图 7 所示.

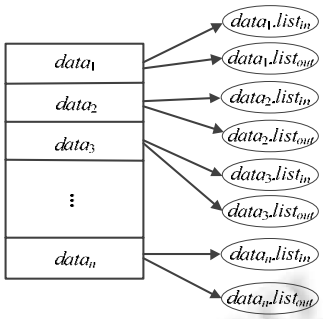


Fig.7 Structure of ASDataIndex

图 7 索引 ASDataIndex 的结构

对于较大规模的语义工作流库,为了提高检索效率,可以不存储($data$,$data.list_{in}$,$data.list_{out}$),而是将($data.hashcode$,$data.list_{in}.pointer$,$data.list_{out}.pointer$)存储到 B+树中[20].其中,$data.hashcode$ 是 $data$ 的哈希值,用作 B+树的键值;$data.list_{in}.pointer$,$data.list_{out}.pointer$ 分别指向 $data.list_{in}$,$data.list_{out}$ 的位置.

索引 ASDataIndex 的建立过程如算法 1 所示.

算法 1. 锚集合数据索引 ASDataIndex 构造算法.

输入:工作流 stream 库 STB.

输出:锚集合数据索引 $asDataIndex$.

construct(STB)

```
1.  asDataIndex=∅
2.  if STB==∅ then
3.     return asDataIndex;
4.  end if
5.  for each St∈STB do
6.     AS_in=computeAS_in(St), AS_out=computeAS_out(St);
7.     for each ad∈AS_in do
8.        if asDataIndex does not contain ad then   /* 如果 asDataIndex 不含 ad,则加入 ad */
9.           asDataIndex.add(ad);
10.       end if
11.       ad.list_in.add(St);
12.    end for
13.    for each ad∈AS_out do
14.       if asDataIndex does not contain ad then
15.          asDataIndex.add(ad);
16.       end if
17.       ad.list_out.add(St);
18.    end for
```

19. **end for**

20. **return** *asDataIndex*.

在算法 1 中，第 4 行的函数 $computeAS_{in}(St)$, $computeAS_{out}(St)$分别用于获取 stream *St* 的输入锚和输出锚集合.易得,只要索引 ASDataIndex 包含数据对象 *ad*,则 $ad.list_{in}$ 和 $ad.list_{out}$ 其中之一必不为空.算法 1 把 stream 库中 stream 的锚集合中的每个数据对象 *ad* 都加入到索引 ASDataIndex 中,这一过程的时间复杂度为 $O(nm)$.由于可能存在重复的数据对象,故空间复杂度的上限为 $O(nm)$,其中,$n=|STB|$,m 为 stream 库中的 stream 的最大数据对象数量.

针对某个查询 stream,为了获得与之匹配的 stream,设计结合索引 ASDataIndex 和 stream 匹配规则的算法对 stream 库进行过滤.过滤算法的流程如图 8 所示.

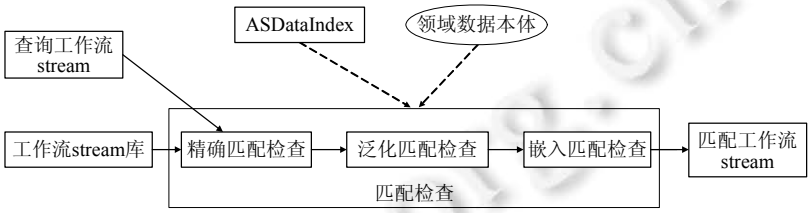


Fig.8 Process of filtering matching streams of the query stream

图 8 过滤某查询 stream 的匹配 stream 的过程

针对查询 stream St_q,对于 stream 库中的每一个 stream *St*,首先,借助锚集合数据索引过滤算法依次检查 *St* 与 St_q 是否满足精确匹配、泛化匹配或嵌入匹配条件.如果满足当前的匹配条件,则保存 *St* 并停止针对 *St* 的检查;否则,检查 *St* 与 St_q 之间是否满足下一种匹配条件.若 3 种匹配条件均不满足,则丢弃 *St*,继续检查下一个 stream.算法伪代码如算法 2 所示.

算法 2. 结合锚集合数据索引和 stream 匹配规则的匹配 stream 过滤算法.

输入:stream 库 *STB*,查询 stream St_q,索引 *asDataIndex*,数据本体 *DataOnto*.

输出:与 St_q 满足精确、泛化或嵌入匹配的 stream 集合 *RS*.

computeMSt(*STB*,St_q,*asDataIndex*,*DataOnto*)

1. $RS=\varnothing$
2. $A_1=computeAS_{in}(St_q)$, $B_1=computeAS_{out}(St_q)$; /* 获取 St_q 的输入锚集合 A_1,输出锚集合 B_1 */
3. get the set *RS* of streams accurately matching St_q in *STB* with *asDataIndex*, A_1, B_1; /* 精确匹配 */
4. **if** $RS\neq\varnothing$ **then return** *RS*;
5. **else** get *RS* of streams generalized matching St_q in *STB* with *asDataIndex*, A_1, B_1; /* 泛化匹配 */
6. **if** $RS\neq\varnothing$ **then return** *RS*;
7. **else** /* 嵌入匹配 */
8. $RS_{in}=STB$, $RS_{out}=STB$;
9. **for** each $ad\in A_1$ **do**
10. $S_1=computeSpecCSet(ad,IN)$;
11. $RS_{in}=RS_{in}\cap(S_1\cup\{ad.list_{in}\})$;
12. **end for**
13. **if** $RS_{in}==\varnothing$ **then return** $\varnothing$;
14. **else**
15. **for** each $ad\in B_1$ **do**
16. $S_2=computeSpecCSet(ad,OUT)$;

17. $RS_{out}=RS_{out}\cap(S_2\cup\{ad.list_{out}\})$;
18. **end for**
19. **end if**
20. $RS=RS_{in}\cap RS_{out}$, **return** RS;
21. **end if**
22. **end if**

函数 *computeSpecCSet*(*ad*,*flag*):

输入:数据对象 *ad*,数据对象本体 *DataOnto*.

输出:包含 *ad* 的一步特化数据对象的 stream 集合 *SP*.

1. $SP=\varnothing$;
2. *Sub*=*oneStepSpecCSet*(*ad*,*DataOnto*); /* 在 *DataOnto* 中获取 *ad* 的一步特化数据对象集 */
3. **if** *flag*=*IN* **then**
4. **for each** $ad_1\in Sub$ **do**
5. **if** *asDataIndex* contains ad_1 **and** $ad_1.list_{in}\neq\varnothing$ **then** $SP=SP\cup ad_1.list_{in}$;
6. **else continue**;
7. **end if**
8. **end for**
9. **else** /* *flag*=*OUT* */
10. **for each** $ad_1\in Sub$ **do**
11. **if** *asDataIndex* contains ad_1 **and** $ad_1.list_{out}\neq\varnothing$ **then** $SP=SP\cup ad_1.list_{out}$;
12. $SP=SP\cup ad_1.list_{out}$;
13. **else continue**;
14. **end if**
15. **end for**
16. **end if**
17. **retrun** *SP*.

算法 2 的第 3 行基于 *asDataIndex*,通过计算 St_q 的输入/输出锚集合中每个数据对象 *ad* 对应的 stream 集合 $ad.list_{in}$ 或 $ad.list_{out}$ 的交集来计算与 St_q 精确匹配的 stream 集合.第 5 行用于检索与 St_q 泛化匹配的 stream 集合;先基于 *asDataIndex* 从 *STB* 中过滤出包含 St_q 的输入/输出锚集合中每个数据对象或其一步特化数据对象的候选 stream 集,然后再使用改编的 Ullmann 算法[18]来检验候选 stream 与 St_q 是否满足泛化匹配.第 7 行~第 20 行用于检索与 St_q 嵌入匹配的 stream 集.其中,RS_{in} 是包含输入锚集合的每个数据对象或其一步特化数据对象的 stream 集,RS_{out} 是包含输出锚集合的每个数据对象或其一步特化数据对象的 stream 集.第 10 行、第 16 行的函数 *computeSpecCSet*(*ad*,*flag*)用于获取包含 *ad* 的一步特化数据对象的 stream 集合.$flag\in\{IN,OUT\}$用于指明 *ad* 来自输入锚集合还是输出锚集合.执行算法 2,可以得到与 St_q 精确匹配、泛化匹配或嵌入匹配的 stream 集合.

函数 *computeSpecCSet*(*ad*,*flag*)的主要计算是第 2 行的求一步特化数据对象集操作和第 5 行的求并集操作.由于数据对象本体 *DataOnto* 形式上为一棵多叉树,第 2 行操作的最差情况是需要遍历这棵树.假设 *DataOnto* 对应的多叉树有 k 个节点,则第 2 行的时间复杂度的上限为 $O(k)$,空间复杂度的上限为 $O(k)$.假设 *ad* 的一步特化概念的个数为 p,则第 5 行求并集的时间复杂度为 $O(p)$.因为 $p\leqslant k$,从而函数 *computeSpecCSet*(*ad*,*flag*)的时间复杂度为 $O(k)$.对于 *STB* 中的每一个 stream 和查询 stream St_q 的每一个数据对象,算法 2 都可能执行 1 次函数 *computeSpecCSet*(*ad*,*flag*).假设 stream 库 *STB* 规模为 n,即 $n=|STB|$,而查询 stream St_q 的最大数据对象的数量为 m,则算法 2 的时间复杂度的上限为 $O(nmk)$,空间复杂度的上限为 $O(nmk)$.

2.1.3 工作流 stream 的行为相似性

执行算法 2 可能得到多个候选匹配 stream.为了确定最佳匹配 stream,根据它们与查询 stream 的相似性对其进行排序是一个好方法.本文使用 stream 的行为相似性来表示 stream 之间的相似性.使用行为特征——任务紧邻关系集合[9]刻画 stream 的执行行为,然后,基于两个 stream 的任务紧邻关系集合的相似性来表示 stream 之间的行为相似性.

定义 10(任务紧邻关系). 假定 TrS 是语义工作流 $SW=(N,E,S,\tau)$的所有可能轨迹的集合,$\langle a,b\rangle$为 SW 的一个任务紧邻关系 TAR,其中,$a,b\in N_T,N_T$ 是任务节点的集合,当且仅当存在一个轨迹 $trace=\langle t_1,t_2,t_3,...,t_i,t_{i+1},...,t_n\rangle(i\in\{1,2,...,n-1\})$满足 $trace\in TrS,t_i=a$ 并且 $t_{i+1}=b.SW$ 的所有 TAR 的集合记为 SW 的任务紧邻关系集(TAR set,简称 TARS).

由于 stream 的图规模较小,可以通过遍历它的语义标注有向图来得到它的 TAR 集.例如在图 3 中,stream S_1,S_2,S_3,S_4 的 TAR 集均可容易得到.

定义 11(任务紧邻关系的相似性). 对于语义工作流 $SW_1=(N_1,E_1,S_1,\tau),SW_2=(N_2,E_2,S_2,\tau)$,任务节点 $a,b\in N_{T1};c,d\in N_{T2}$,其中,$N_{T1},N_{T2}$ 分别是 SW_1,SW_2 的任务节点集合;$tar_1=\langle a,b\rangle,tar_2=\langle c,d\rangle$分别是 SW_1,SW_2 的任务紧邻关系;C_1,C_2,C_3,C_4 分别是 a,b,c,d 的语义描述对应于领域任务本体的语义概念.如果 $C_1\sqsubseteq C_3,C_2\sqsubseteq C_4$,则 $sim(tar_1,tar_2)=1$;否则,

$$sim(tar_1,tar_2)=(sim(C_1,C_3)+sim(C_2,C_4))/2 \tag{2}$$

其中,$sim(C_1,C_3),sim(C_2,C_4)$的计算方法如公式(1).

定义 11 中的 $C_1\sqsubseteq C_3,C_2\sqsubseteq C_4$,而不是 $C_1\sqsubseteq^* C_3,C_2\sqsubseteq^* C_4$ 是为了获取更多的行为相似 stream.根据领域专家意见,设定了任务紧邻关系的相似性阈值 $cut\text{-}off_1$.当 $sim(tar_1,tar_2)\geqslant cut\text{-}off_1$ 时,可认为 tar_1,tar_2 相同,否则不相同.文献[21]在计算过程模型中功能(function)的标签间的语义相似性时,使用了同义词相似性方法,通过实验获得最优相似性阈值 0.89.在引入领域知识的前提下,语义概念之间的相关性增大,本文参照文献[21]的最优相似性阈值确定方法并做调整,取相似性阈值 $cut\text{-}off_1=0.8$.

定义 12(工作流 stream 的行为相似性). 对于工作流 stream St_1,St_2,如果 St_2,St_1 是精确匹配或者泛化匹配,即 $St_2\sqsubseteq^* St_1$,则 St_1,St_2 的行为相似性 $sim_b(St_1,St_2)=1$.如果 St_2,St_1 是嵌入匹配,设 St_1 的 TAR 集为 TS_1,St_2 的 TAR 集为 TS_2,则 St_1,St_2 的行为相似性为

$$sim_b(St_1,St_2)=|TS_1\cap TS_2|/|TS_1\cup TS_2| \tag{3}$$

否则,$sim_b(St_1,St_2)=0$.

由于为 TAR 的相似性设定了相似性阈值,故 $TS_1\cap TS_2$ 和 $TS_1\cup TS_2$ 容易得到.计算 $sim_b(St_1,St_2)$还需要获得 TS_1,TS_2 的最优匹配,该问题可以使用 Kuhn-Munkres[22]算法来解决.由于 St_1,St_2 的图规模较小,故获取最优匹配的计算量也较小.

2.1.4 结合变更请求检索相似的匹配工作流 stream

在语义工作流重用过程中,查询中的特殊需求一般为在检索工作流中需要包含或不需要包含的任务或数据对象集合.它可被转换成对检索工作流的变更要求,即需要在检索工作流中删除或加入的任务节点和数据对象节点的集合.变更请求可形式化表示为 $chReq=Delete_T\wedge Delete_D\wedge Add_T\wedge Add_D$.其中,$Delete_T=(\neg task_1\wedge\neg task_2\wedge\ldots\wedge\neg task_m)$,$Delete_D=(\neg data_1\wedge\neg data_2\wedge\ldots\wedge\neg data_q)$, $Add_T=(task_1'\wedge task_2'\wedge...\wedge task_n')$, $Add_D=(data_1'\wedge data_2'\wedge...\wedge data_p')$. $task_1,task_2,\ldots,task_m$ 是需要删除的任务节点; $task_1',task_2',...,task_n'$ 是需要加入的任务节点;$data_1,data_2,\ldots,data_q$ 是需要删除的数据对象; $data_1',data_2',...,data_p'$ 是需要加入的数据对象.

目前,本修正算法仅支持需要在检索工作流中删除或加入数据对象的变更请求,即 $chReq=Delete_D\wedge Add_D$.为计算方便,分别把 $Delete_D,Add_D$ 转化为需要删除的数据对象集 DE、需要加入的数据对象集 AD.

对于检索语义工作流中的每个查询 stream,由算法 2 可以获得与之匹配的 stream 集.本节基于变更请求和 stream 行为相似性对匹配 stream 集验证,从中检索出需替换的 stream 和最符合变更请求 $chReq$ 并与该 stream 足够行为相似的匹配 stream.“需替换的查询 stream”是指该 stream 包含当前 $chReq$ 的 DE 中的数据对象,或者针

对该 stream 可以检索到最符合变更请求 *chReq* 且与它足够行为相似的匹配 stream.“最符合变更请求 *chReq*”指的是 stream 不包含 *chReq* 的 *DE* 中的数据对象,并且包含 *chReq* 的 *AD* 中的数据对象的数量最多.

检索算法的流程如图 9 所示.

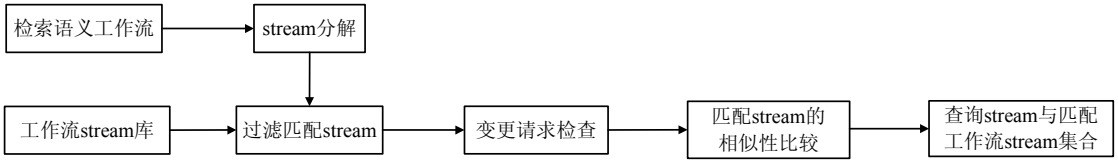


Fig.9 Process of retrieving behaviorally similar matching streams most coincident to change request

图 9 检索最符合变更请求且行为相似的匹配 stream 的过程

检索算法首先分解检索语义工作流 *SW*,得到查询 stream 集合 WS_0;然后,对于 WS_0 中的每个查询 stream St_q,使用算法 2 过滤 stream 库得到匹配 stream 集 WS_1;接着,从 WS_0 和 WS_1 中检索出需替换的 stream St_a 和最符合变更请求 *chReq* 并与 St_a 足够行为相似的匹配 stream St_m,组成序偶$\langle St_a,St_m\rangle$;最后,得到序偶$\langle St_a,St_m\rangle$的集合.

检索算法的伪代码如算法 3 所示.

算法 3. 获取检索语义工作流中的需替换的 stream 及其对应的匹配 stream.

输入:工作流 stream 库 *STB*,检索语义工作流 *SW*,索引 *asDataIndex*,变更请求 *chReq*.

输出:*SW* 中的需替换的 stream 及其对应的匹配 stream 映射的集合 map_s.

1. $WS_0=\varnothing$, $WS_1=\varnothing$, $map_s=\varnothing$;
2. $WS_0=computeST(SW)$, $DE=getDE(chReq)$, $AD=getAD(chReq)$;
3. **while** $WS_0\neq\varnothing$ **do**
4. select a stream $St_q\in WS_0$;
5. $WS_1=computeMSt(STB,St_q,asDataIndex,DataOnto)$; /* 运行算法 2,获取 St_q 的匹配 stream 集合 */
6. $WS_2=\varnothing$;
7. **for** each $St_1\in WS_1$ **do**
8. $D_1=computeDS(St_1)$; /* 获取 St_1 的数据对象集 */
9. **if** $D_1\cap DE==\varnothing$ **then** $WS_2=WS_2\cup\{St_1\}$;
10. **end if**
11. **end for**
12. $priQueue=getPriQueue(WS_2,AD)$;
13. **while** $priQueue\neq\varnothing$ **do**
14. $St_2=priQueue.get(0)$, $D_2=computeDS(St_q)$, $D_3=computeDS(St_2)$;
15. **if** $D_2\cap DE\neq\varnothing\vee D_3\cap AD\neq\varnothing$ **and** $sim_b(St_q,St_2)\geqslant cut\text{-}off_2$ **then**
$map_s.put(\langle St_q,St_2\rangle)$, $WS_0=WS_0-St_q$, $AD=AD-D_3$, **break**;
16. $priQueue.pop()$;
17. **end if**
18. **end while**
19. $WS_0=WS_0-St_q$,
20. **end while**
21. **return** map_s.

在算法 3 中,第 2 行的函数 $computeST(SW)$,$getDE(chReq)$,$getAD(chReq)$分别用于获取 *SW* 的 stream 集合、*chReq* 的 *DE* 和 *AD*.第 7 行~第 11 行用于获取与 *DE* 不矛盾的 stream 集合 WS_2.第 12 行用于将 WS_2 中的 stream 按包含 *AD* 中的数据对象的数量从多到少进行排序,得到优先队列 *priQueue*.第 13 行~第 18 行用于获取需替换

的 stream St_q 与 *priQueue* 中的对应 stream 的映射集合 map_s.第 15 行保证只有至少包含 *DE* 中的一个数据对象的查询 stream St_q 或 *AD* 中一个数据对象的匹配 stream St_2 才进行行为相似性检验.此时与 St_q 行为相似性足够高的 St_2 被用于替换 St_q,目的是保证替换操作对修正 *SW* 的缺陷有促进作用.在 stream 替换阶段,可以根据 map_s 中的匹配 stream 对,使用匹配 stream 来替换对应的查询 stream.如果运行算法 3 不能得到这样的映射集合 map_s,则可以适当放松约束条件获取一个可以接受的映射集合 map_s,如减小行为相似性阈值 $cut\text{-}off_2$ 或放松变更请求的约束等.

算法 3 的主要计算在于第 5 行的匹配 stream 检索、第 12 行的优先队列生成和第 13 行~第 18 行的获取映射集合 map_s.其中,第 5 行的时间复杂度的上限为 $O(nmk)$,第 13 行的时间复杂度的上限为 $O(n^2)$,第 13 行~第 18 行的时间复杂度的上限为 $O(n)$.设检索语义工作流 *SW* 中的 stream 数量为 q,则算法 3 的时间复杂度为 $O(qnmk+qn^2+qn)=O(qnmk+qn^2)$.由于第 5 行的空间复杂度的上限为 $O(nmk)$、第 12 行的空间复杂度的上限 $O(n)$、第 13 行~第 18 行的空间复杂度为 $O(n)$,故算法 3 的空间复杂度的上限为 $O(qnmk)$.

为了便于评判检索到的查询 stream 与匹配 stream 之间是否行为相似,本文设置了一个行为相似性阈值 $cut\text{-}off_2$.当查询 stream 与匹配的 stream 行为相似性大于 $cut\text{-}off_2$ 时,认为二者行为相似,否则不相似.文献[14]提出:可以根据相似性算法的检索性能指标之一——*F* 测度(*F*-measure)的最优值确定相似性阈值的最优值,指出,基于有代表性轨迹的集合的行为相似性算法的最优相似性阈值在 0.70 附近.与文献[14]不同,本文在已知两个 stream 匹配的前提下确定 $cut\text{-}off_2$ 的数值.根据实际情况并参考领域专家意见,本文设定行为相似性阈值 $cut\text{-}off_2 \geqslant 0.55$.暂取 $cut\text{-}off_2=0.55$,可以根据实际情况对其进行调整.例如,认为行为相似性为 0.65 的一对匹配 stream、行为相似性为 0.85 的一对匹配 stream 均为行为相似,但是两对匹配 stream 的相似程度不同.使用类似的方法确定匹配 stream 间的结构相似性阈值 $cut\text{-}off_3 \geqslant 0.55$,暂取 $cut\text{-}off_3=0.55$.

2.2 修正检索语义工作流

执行算法 3,可以得到查询 stream St_a 与其对应匹配 stream St_m 的映射集合 map_s.之后,检索语义工作流 *SW* 的修正由每个 St_m 替换对应的 St_a 完成.具体操作:先在 *SW* 中删除 St_a,然后将 St_m 插入 *SW* 中.插入 St_m 的位置是 *SW* 中 St_a 的最后一个顺序节点处[2].

假定图 4 中的语义工作流 SW_1 是基于结构或行为相似性检索到的最相似语义工作流,变更请求 $chReq=\neg Cheddar \wedge Vermicelli$,故 SW_1 的 stream S_4 是缺陷 stream.即需要在 SW_1 中需要删除语义描述为 *Cheddar* 的数据对象节点,加入语义描述为 *Vermicelli* 的数据对象节点.现在对 SW_1 进行修正.Müller 等人[2]的方法完成修正的前提是在 stream 库中存在与查询 stream 结构相似的 stream.假设目前在 stream 库中存在与 S_4 结构相似性最高的 stream St_4,如图 10 所示.由 Müller 等人[2]的方法得 S_4,St_4 的结构相似性 $sim_s(S_4,St_4)=0.26<cut\text{-}off_3$,从而 Müller 等人[2]的方法在相似的匹配 stream 检索时会忽略 St_4,所以会检索不到合适的匹配 stream,因而无法修正工作流 SW_1.

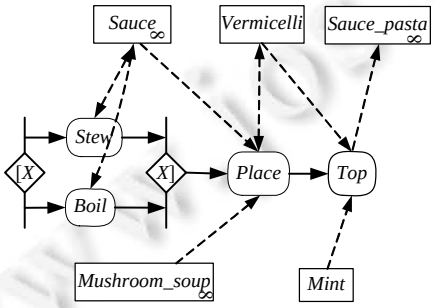


Fig.10 Workflow stream St_4

图 10 工作流 stream St_4

由定义 8 得,stream St_4,S_4 是嵌入匹配类型,故可以用 St_4 替换 S_4.为了简化,此处以语义描述指代所属的任务节点.由文献[9]得,S_4 的任务紧邻关系集 TS_1={⟨*Simmer*,*Place*⟩,⟨*Place*,*Pour*⟩},St_4 的任务紧邻关系集 TS_2={⟨*Stew*,*Place*⟩,⟨*Boil*,*Place*⟩,⟨*Place*,*Top*⟩}.由定义 11 得,sim(⟨*Simmer*,*Place*⟩,⟨*Stew*,*Place*⟩)=0.85>0.8,故可视⟨*Simmer*,*Place*⟩与⟨*Stew*,*Place*⟩等价.从而由定义 12 得,$TS_1 \cap TS_2$={⟨*Simmer*,*Place*⟩,⟨*Place*,*Pour*⟩},$TS_1 \cup TS_2$={⟨*Simmer*,*Place*⟩,⟨*Boil*,*Place*⟩,⟨*Place*,*Top*⟩}.所以 $sim_b(S_4,St_4)=|TS_1 \cap TS_2|/|TS_1 \cup TS_2|=0.67>cut\text{-}off_2$,故当前的 St_4 是行为相似的匹配 stream.于是,可以使用 St_4 替换 S_4,对语义工作流 SW_1 进行修正.

使用 stream St_4 替换 stream S_4 的过程如下:

1) 从语义工作流 SW_1 中删除工作流 stream S_4(如图 11 所示);
2) 将工作流 stream St_4 加入到语义工作流 SW_1 中(如图 12 所示).

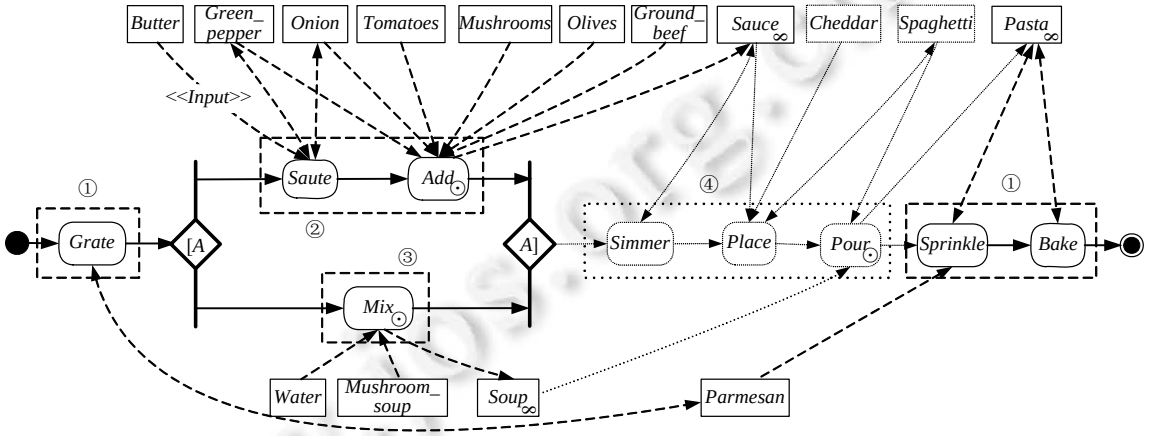


Fig.11 Remove stream S_4 from SW_1

图 11 从 SW_1 中删除 S_4

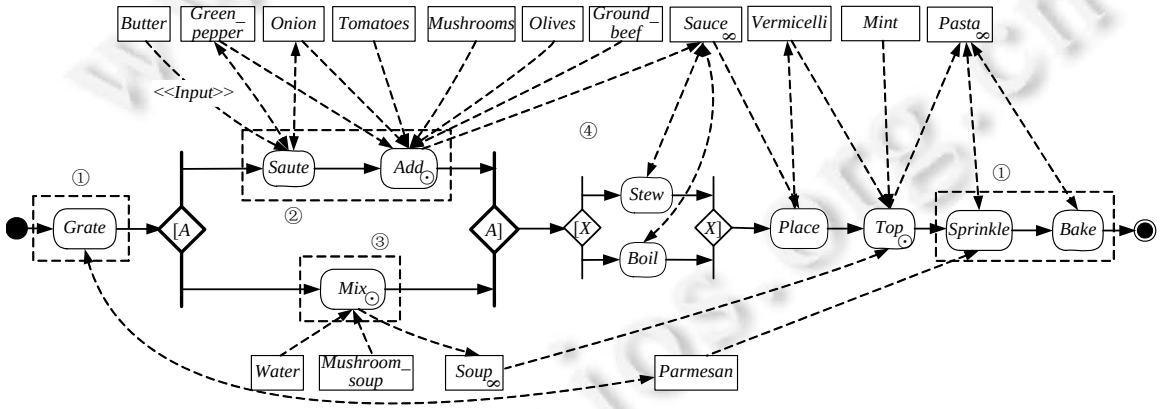


Fig.12 Add stream St_4 to SW_1

图 12 把 St_4 加入 SW_1 中

使用行为最相似的匹配 stream 替换查询 stream 可能带来检索语义工作流结构上的改变,但只要修正后的语义工作流能够满足数据流连接的约束,均认为修正是可行的.传统工作流的任务节点不包含输入/输出数据对象,无法根据输入/输出数据对象对它们进行分解得到 stream 集合.这样,对不能直接复用的传统工作流无法进行修正.在工作流变更时,为了能够重用现有的传统工作流中有意义的片段——stream,对它们进行分解,得到 stream 库是必要的.可用的分解方法有结合人类专家的经验确定分割点进行分解、依据有意义的最小控制流块

进行分解等.这样,在重用传统工作流时,可以通过检索行为相似的 stream 来修正检索工作流.

3 实验与结果分析

3.1 实验设计

本文的实验基于如下环境:Intel Core(TM) CPU2.2 GHz,8.0GB RAM,Windows7 64 位操作系统,JDK1.6 环境的计算机.目前,专用的语义工作流的实验数据集还较少.本文的实验数据集选自 WikiTaaable(http://wikitaaable.loria.fr)的 Recipe 和 Recipesource(http://www.recipesource.com),其中,共有 1 000 多个意大利面食(pasta)食谱.实验中的领域任务本体 *TaskOnto* 来自 WikiTaaable 的 Culinary actions 本体,有 100 个左右节点;领域数据对象本体 *DataOnto* 来自 WikiTaaable 的 Food 本体,有 200 个左右节点.WikiTaaable,Recipesource 及其中的本体是 ICCBR 会议为计算机烹饪比赛(computer cooking contest,简称 CCC)建立的.

本文从意大利面食食谱数据集中选取 100 个食谱[14],根据定义 1,将食谱的烹饪说明(cooking instruction)表示成如图 1 的语义工作流[1,14],组成规模为 100 的烹饪语义工作流库 SWB_0.本实验从 SWB_0 中随机选取 50 个语义工作流,组成测试语义工作流库 SWB_1.SWB_1 的基本结构特征见表 1.

Table 1 Basic characteristics of the case base of culinary semantic workflows

表 1 测试烹饪语义工作流库的基本特征

数据集	平均任务节点数	平均数据节点数	平均路由节点数	平均控制边数	平均并发分支数
烹饪语义工作流库	7.8	12.3	3.2	14.6	2.5

同时,请 3 位烹饪专家组成领域专家组来评价检索到的匹配工作流 stream 和修正语义工作流的质量.

3.2 本文的修正方法与基于结构特征的修正方法比较

首先开展实验 1:比较基于行为特征的 stream 相似性方法和基于结构特征的 stream 相似性方法[2]对相似的匹配 stream 的检索性能.从 SWB_1 任选 10 个工作流组成测试工作流集 QSW_1.对每一个测试语义工作流,任选它的一个 stream 记录下来.对该 stream 做如下改变:如果为顺序结构,则为它的某个任务节点添加包含一个语义描述相似的任务节点的并发和选择分支;如果为并发或选择结构,则任选其中的两个任务节点,将其改变为语义描述相似的任务节点.这样共记录 10 个 stream 并组成检索 stream 集 *QST*,同时可以得到 20 个改变后的语义工作流.这 20 个改变后的语义工作流与余下的 40 个语义工作流一起组成规模为 60 的新语义工作流库 SWB_2.对 SWB_2 中的每个语义工作流进行分解,共得到 215 个 stream,组成工作流 stream 库 STB_1.

针对检索 stream 集 *QST*,在 STB_1 中检索相似的匹配 stream.具体如下:对于 *QST* 中的每个 stream(称为查询 stream),首先,在 STB_1 中检索匹配 stream 集合 MS_1;然后,在 MS_1 中检索行为特征相似的 stream.

对于定义 12 的 stream 的行为相似性,采用基于相似性的检索(similarity based retrieval)方法执行上述检索过程.记录每个查询 stream 的检索结果集中的相似匹配 stream 的数量,并对所有查询 stream 的相似匹配 stream 的数量求均值.使用基于结构特征的修正方法[2]中的 stream 相似性方法执行相同的检索过程.检索结果见表 2.

Table 2 Comparisions of retrieval results of similar matching workflow streams

表 2 相似的匹配工作流 stream 的检索结果比较

算法	基于结构特征的相似性	基于行为特征的相似性
相似的匹配工作流 stream 的平均数量	2.1	3.3

由表 2 得,基于行为特征的相似方法的检索结果集中,匹配 stream 的平均数量高于基于结构特征的相似方法[2].这是因为基于 stream 行为特征的方法不但检索出了结构相似的匹配 stream,而且检索出了行为相似的匹配 stream.这样可以在结构相似的匹配 stream 不存在的情况下,选用行为相似的匹配 stream 来完成语义工作流修正任务.

同时,请领域专家组评价检索到的 stream 的质量,分别绘制两种相似性方法对应的 *P-R* 曲线.如图 13 所示.

P-R(precision-recall)曲线是评价检索系统的整体检索性能的方法之一.整体检索性能较好的检索系统的 *P-R* 曲线全部或绝大部分处于整体检索性能一般的检索系统.

由图 13 可以得出,基于行为特征相似性的匹配 stream 检索方法的 *P-R* 曲线整体处于基于结构特征相似性的检索方法[2]的 *P-R* 曲线上方.这说明基于行为特征相似性的匹配 stream 检索方法的检索性能要好于基于结构特征相似性的检索方法[2].

接下来开展实验 2:检验本文的基于 stream 行为特征的工作流修正方法是否执行了变更要求.从测试工作流集 SWB_2 中任选 10 个工作流组成测试工作流集 QSW_2.针对 QSW_2 中的每个测试工作流 TW_i,首先删除 1~2 个数据对象节点,并将其 1~2 个数据对象节点的语义描述替换为相似的语义描述,得到语义工作流 TW_i'.在 SWB_2 中,用 TW_i' 替换 TW_i.然后,对 QSW_2 中的每个测试工作流 TW_i,使用语义工作流的结构相似性方法[1]在 SWB_2 中检索一个相似的语义工作流 RW_i,以使得可以通过执行一组删除(delete)和加入(add)数据对象节点的操作将 RW_i 转变为 TW_i[2].这些删除和加入操作由手工得出,作为工作流修正时的变更要求 *chReq*.然后,对于每个语义工作流 RW_i 及其对应的 *chReq*,执行本文的工作流修正算法,可以得到 10 个修正语义工作流.分析 10 个修正语义工作流,并请领域专家组对它们进行评价.结果表明,测试语义工作流均完成了修正过程,并且与变更请求 *chReq* 基本一致.

最后开展实验 3:比较基于 stream 行为特征的修正方法和基于 stream 结构特征的修正方法[2]得到的修正工作流的质量.将实验 2 中的测试工作流集 QSW_2 中的每个测试工作流 TW_i 与其对应的修正语义工作流 RW_i 混在一起,这样得到包含 20 个工作流的工作流集合 *RWS*.在事先不告诉每个工作流是测试工作流还是修正语义工作流的前提下,请 3 位领域专家使用 Likert 评分方法(1~5 分)对 *RWS* 中的每个工作流的合理性进行评分.然后将每个测试工作流 TW_i 和对应修正语义工作流 RW_i 组成一对(TW_i,RW_i),这样共得到 30 对.比较每对(TW_i,RW_i)中两个工作流评分的高低.同时,对测试工作流集 QSW_2 中的每个语义工作流 TW_i 及其对应的变更要求 *chReq*,使用基于 stream 结构特征的修正方法[2]执行工作流修正操作.对得到的修正工作流进行相同的评分过程.结果如图 14 所示.

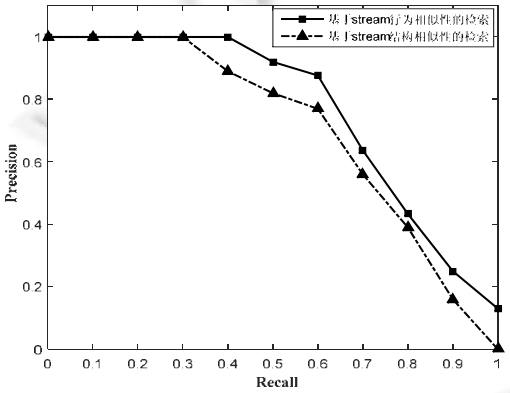


Fig.13 Comparisions of retrieval performances on workflow streams

图 13 工作流 stream 的检索性能比较

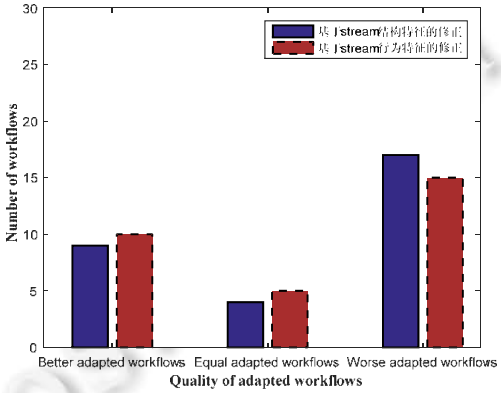


Fig.14 Comparisions of qualities of adapted semantic workflow

图 14 修正语义工作流的质量比较

图 14 的横坐标分别为在语义工作流修正结束后,领域专家对修正语义工作流的评分高于、低于对测试语义工作流的评分和领域专家对二者的评分相等这 3 种情况;纵坐标为 3 种情况对应的修正工作流数量.

由图 14 可以看出,对于基于 stream 行为特征的修正方法,共有 15 个修正语义工作流的评分不低于(高于或等于)测试工作流,比例为 50%.而基于 stream 结构特征的修正方法[2]共有 13 个修正语义工作流的评分不低于(高于或等于)测试工作流,比例约为 43%.基于行为特征的修正方法的评分比基于结构特征的修正方法高出 7%.从而与基于 stream 结构特征的修正方法[2]相比,基于 stream 行为特征的修正方法得到了整体质量更好的修正语

义工作流集合.

从以上 3 个实验可得:与基于 stream 结构特征的修正方法[2]相比,基于 stream 行为特征的修正方法提高了修正算法的适应能力,较好地改善了修正语义工作流的质量.同时也可以得出:基于 stream 结构特征的修正方法[2]更适合于科学工作流的修正,而本文的基于 stream 行为特征的修正方法更适合于业务工作流的修正.

4 总结与展望

本文提出了一种不需要修正知识、基于工作流 stream 行为特征的语义工作流修正算法,该算法适用的前提是具有一定规模的语义工作流库.该算法以引入领域知识的块结构化语义工作流为研究对象,使用任务紧邻关系集刻画工作流 stream 的执行行为.结合领域数据知识构造了工作流 stream 库的锚集合数据索引 ASDataIndex.针对检索语义工作流中的每个查询工作流 stream,先基于 ASDataIndex 和工作流 stream 匹配规则对工作流 stream 库进行过滤,得到候选匹配工作流 stream 集;然后,基于行为特征的相似性和变更请求对候选工作流 stream 集进行验证,得到与当前变更请求一致程度最高和相似的匹配工作流 stream;接着,使用检索到的匹配工作流 stream 替换原查询工作流 stream,逐步对检索语义工作流进行修正;最后得到修正语义工作流.实验结果表明,与基于工作流 stream 结构特征的修正算法[2]相比,本文的基于工作流 stream 行为特征的语义工作流修正算法可以在不存在结构足够相似的匹配工作流 stream 的情况下,使用行为足够相似的匹配工作流 stream 完成检索工作流修正;并且得到了整体质量更好的修正语义工作流集.本文的工作流修正算法为业务过程管理人员为适应新业务需求而进行的工作流变更提供了具有较好参考价值的修正语义工作流,对提高实际业务工作流管理中的工作流重用的效率和质量有较大帮助.

在未来的工作中,将致力于将本文的工作流修正算法应用于其他领域的业务流程修正;研究更高效的工作流 stream 的行为特征刻画方法和优化的修正操作,进一步提高基于行为特征的语义工作流修正算法的效率.

References:

[1] Bergmann R, Gil Y. Similarity assessment and efficient retrieval of semantic workflows. Information Systems, 2014,40(1): 115−127.

[2] Müller G, Bergmann R. Workflow streams: A means for compositional adaptation in process-oriented CBR. In: Proc. of the Case-Based Reasoning Research and Development. Springer Int'l Publishing, 2014. 315−329.

[3] Mantaras RLD, Mcsherry D, Bridge D, *et al*. Retrieval, reuse, revision and retention in case-based reasoning. Knowledge Engineering Review, 2005,20(3):215−240.

[4] Minor M, Bergmann R, Görg S, *et al*. Towards case-based adaptation of workflows. In: Proc. of the Case-Based Reasoning Research and Development. Berlin, Heidelberg: Springer-Verlag, 2010. 421−435.

[5] Minor M, Bergmann R, Görg S, *et al*. Adaptation of cooking instructions following the workflow paradigm. In: Proc. of the ICCBR 2010 Workshop Proc. 2010. 199−208.

[6] Müller G, Bergmann R. Generalization of workflows in process-oriented case-based reasoning. In: Proc. of the Int'l Flairs Conf. 2015. 391−396.

[7] Müller G, Bergmann R. Learning and applying adaptation operators in process-oriented case-based reasoning. In: Proc. of the Case-Based Reasoning Research and Development. Springer Int'l Publishing, 2015. 259−274.

[8] Jin T, Wang JM, Wen LJ. Efficient retrieval of similar workflow models based on behavior. In: Proc. of the Web Technologies and Applications. Berlin: Springer-Verlag, 2012. 677−684.

[9] Zha HP, Wang JM, Wen LJ, *et al*. A workflow net similarity measure based on transition adjacency relations. Computers in Industry, 2010,61(5):463−471.

[10] Bae J, Liu L, Caverlee J, *et al*. Development of distance measures for process mining, discovery and integration. Int'l Journal of Web Services Research (IJWSR), 2007,4(4):1−17.

[11] Wang JM, HE TF, Wen LJ, *et al*. A behavioral similarity measure between labeled Petri nets based on principal transition sequences. In: Proc. of the Move to Meaningful Internet Systems (OTM 2010). Springer-Verlag, 2010. 394−401.

[12] Dijkman R, Dumas M, Van Dongen B, *et al*. Similarity of business process models: Metrics and evaluation. Information Systems, 2011,36(2):498−516.

[13] Kunze M, Weidlich M, Weske M. Behavioral similarity—A proper metric. In: Rinderle-Ma S, Toumani F, Wolf K, eds. Proc of the 9th Int'l Conf. on Business Process Management (BPM 2010). Berlin, Heidelberg: Springer-Verlag, 2011. 166−181.

[14] Sun JY, Gu TL, Wen LJ, *et al*. Similarity algorithm for semantic workflows used in process-oriented case-based reasoning. Computer Integrated Manufacturing Systems, 2016,22(2):381−394 (in Chinese with English abstract).

[15] Song JF, Wen LJ, Wang JM. A similarity measure for process models based on task occurrence relations. Journal of Computer Research and Development, 2017,54(4):832−843 (in Chinese with English abstract).

[16] Dufour-Lussier V, Leber F, Lieber J, *et al*. Automatic case acquisition from texts for process-oriented case-based reasoning. Information Systems, 2014,40(1):153−167.

[17] Kiepuszewski B, Hofstede AHM, Bussler CJ. On structured workflow modelling. In: Proc. of the Seminal Contributions to Information Systems Engineering. Berlin, Heidelberg: Springer-Verlag, 1999. 241−256.

[18] Ullmann JR. An algorithm for subgraph isomorphism. Journal of the ACM, 1976,23(1):31−42.

[19] Bergmann R, Stromer A. MAC/FAC retrieval of semantic workflows. In: Proc. of the 26th Int'l Florida Artificial Intelligence Research Society Conf. Menlo Park: AAAI, 2013. 357−362.

[20] Jin T, Wang JM, Wu NH, *et al*. Efficient and accurate retrieval of business process models through indexing. LNCS, 2010,6426:402−409.

[21] Dongen B, Dijkman R, Mendling J. Measuring similarity between business process models. In: Proc. of the Int'l Conf. on Advanced Information Systems Engineering. Springer-Verlag, 2008. 450−464.

[22] Munkres J. Algorithms for the assignment and transportation problems. Journal of the Society for Industrial & Applied Mathematics, 1957,5(1):32−38.

附中文参考文献:

[14] 孙晋永,古天龙,闻立杰,钱俊彦.用于面向过程的基于实例推理的语义工作流相似性算法.计算机集成制造系统,2016,22(2):381−394.

[15] 宋金凤,闻立杰,王建民.基于任务发生关系的流程模型相似性度量.计算机研究与发展,2017,54(4):832−843.

孙晋永(1978－),男,山东枣庄人,博士,讲师,CCF 专业会员,主要研究领域为业务过程管理,知识表示与推理.

钱俊彦(1973－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为软件工程,模型检测,程序验证.

古天龙(1964－),男,博士,教授,博士生导师,主要研究领域为软件工程与形式化方法,知识工程,符号推理.

刘华东(1978－),男,博士生,讲师,主要研究领域为知识工程,符号推理.

闻立杰(1977－),男,博士,副教授,博士生导师,主要研究领域为业务数据管理,大过程数据,业务过程管理,工作流技术.