Modeling Industrial Business Processes for Querying and Retrieving Using OWL+SWRL

Suman Roy[1(✉)], Gabriel Silvatici Dayan[2], and V. Devaraja Holla[3]

[1] Optum Global Solutions India Pvt. Ltd. (UnitedHealth Group), Bangalore 560 103, India
Suman.Roy@optum.com

[2] Software Engineering, Instituto Tecnologico de Buenos Aires (ITBA), Buenos Aires, Argentina
gsilvati@itba.edu.ar

[3] Infosys Ltd., # 44 Electronics City, Hosur Road, Bangalore 560 100, India
devaraja_vaderahobli@infosys.com

Abstract. Process modeling forms a core activity in many organizations in which different entities and stakeholders interact for smooth operation and management of enterprises. There have been few work on semantically labeling business processes using OWL-DL that formalize business process structure and query them. However, all these methods suffer from few limitations such as lack of a modular approach of ontology design, no guarantee of a consistent ontology development with TBox and ABox axioms and no provision of combining control flow relations of the main process and its sub-processes. In this work, we propose an approach for labeling and specifying business processes by using hybrid programs which offers modular ontology design, consistent ontology design of each module and unified control flow for process and its sub-processes. This formalism of hybrid programs integrates ontology specified in OWL-DL with SWRL (Semantic Web Rules Language) rules. Further we report on our experimental effort on modeling industrial business processes with this hybrid formalism. We also present a case study of an industrial business process to illustrate our modeling approach which can aid in business knowledge management.

Keywords: Business process management
Business processes modeling · Patterns · Ontology · OWL-DL
SWRL · Querying · Case study

1 Introduction

Hundreds of business process models are developed by enterprises to manage the flow of work through an organization. For a process model one needs to con-

G. S. Dayan—This work was done when the author was an intern with Infoys Ltd, Bangalore during Mar–May'17.

sider the semantics of the meta-model elements when different representations are used, and as well as the terms that describe the model elements. The realization of such a semantic modeling of processes can be achieved by extracting ontology from process models using appropriate semantic web formalisms which can facilitate querying and retrieving process models [12].

Several approaches have been proposed for semantic modeling of processes and subsequent retrieval of them using keywords, or data, or process properties. In one such approach Groener and Staab have proposed an ontology modeling of a process capturing explicitly various hierarchical and ordering relationships between activities in [8]. Moreover, they can query for retrieving process structures with specified control flow characteristics and modalities as well as process traces. However, such a modeling suffers from a few drawbacks which we shall discuss using an example of a process model below.

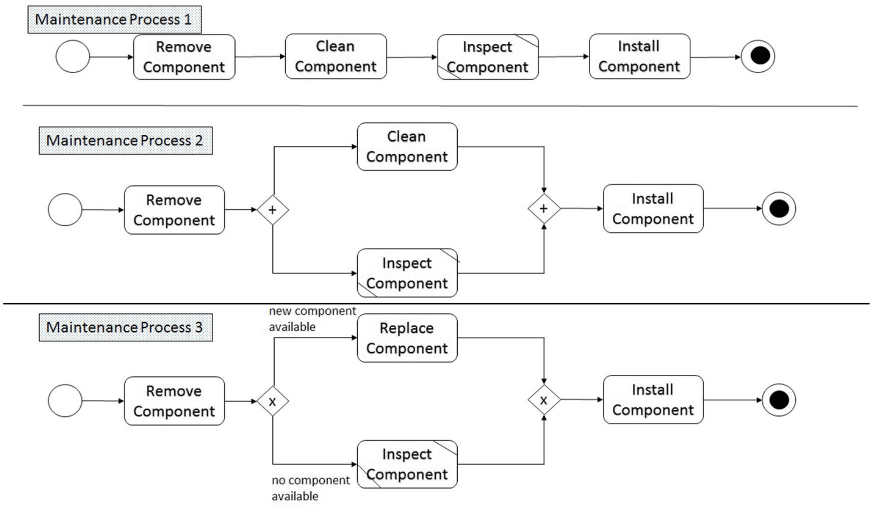


Fig. 1. Different variants of a maintenance process

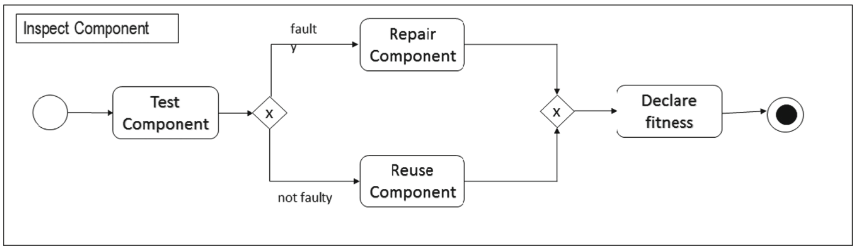


Fig. 2. Inspect component subprocess

A Motivation for New Modeling Framework. In Fig. 1 we describe three maintenance processes, Maintenance Process 1 ($MProcess1$), Maintenance Process 2 ($MProcess2$) and Maintenance Process 3 ($MProcess3$), with three different characteristics. In these process diagrams there are four main activities, 'Remove Component (RC)', 'Clean Component (CC)', 'Replace Component (RpC)' and 'Install Component (IC)'. There is also a sub-process invocation activity, Inspect Component (SPIC) - this sub-process activity can be further decomposed into constituent activities (in Fig. 2), 'Test Component (TC)', 'Repair Component (RepC)', 'Reuse Component (ReuC)' and 'Declare Fitness (DF)'. We assume that the class (node) Activity can be instantiated with individuals in different cases as follows, *actRC* ('Remove Component'), *actCC* ('Clean Component'), *actRpC* ('Replace Component') and *actIC* ('Install Component'), *actTC* ('Test Component'), *actRepC* ('Repair Component'), *actRC* ('Reuse Component'), *actDF* ('Declare Fitness') etc. The sub-process invocation activity is instantiated as *spIC* (stands for 'Inspect Component')[1].

The ontology modeling framework described in [8] would be able to model a OWL-DL ontology out of these processes. However, this framework still requires writing the ontology for the process as a whole. For example, using this approach the ontology for Maintenance Process 2 (its control flow named as $followedBy2$) can be written as:

$$MProcess2 \equiv Start \sqcap = 1 followedBy2.(RC \sqcap \exists followedBy2.(CC \sqcap = 1 followedBy2.(IC \sqcap \exists followedBy2.End)) \sqcap \exists followedBy2.(SPIC \sqcap = 1 followedBy2.(IC \sqcap = 1 followedBy2.End))).$$

However, for a large process writing the whole ontology in one attempt would be tedious, especially in presence of nested gateways. To alleviate this, we aim to capture ontology for each pattern as one module and combine them to obtain the whole ontology. Although any piece of OWL can be easily added to any other OWL ontology just by merging the corresponding triples it is not clear how a OWL ontology can be designed for a single fragment of a process in the framework of [8]. Our pattern-oriented approach of capturing OWL ontology facilitates this. We specify each such pattern in a process as a complex OWL concept using TBox axioms, see Table 1. Also in [8] it is not mentioned how one can write ABox axioms so that the ontology remains consistent. For that we design consistent ontology for each pattern by formulating ABox axioms along with TBox axioms as shown in Table 1[2] and merge them to obtain the consolidated ontology for the process.

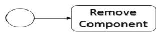
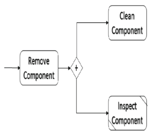
Moreover, the sub-process activity 'Inspect Component' can be specified using OWL-DL as below (with its control flow relation as $followedBy4$) using the approach of [8]:

$$SPIC \equiv Start \sqcap \exists followedBy4.(TC \sqcap \exists followedBy4.(((RepC \sqcap fault) \sqcup (ReuC \sqcap nofault)) \sqcap \exists followedBy4.(DF \sqcap \exists followedBy4.End)))$$

[1] Wlog we assume same individuals for all the three processes.

[2] For brevity of space we specify only a couple of patterns here.

Table 1. A pattern-oriented modeling of maintenance process 2

Patterns	TBox axioms	ABox axioms
	$MProcess2 \equiv Start \sqcap$ $= 1 followedBy2.\mathcal{C}_{RC}$	$Start(start1), Task(actRC),$ $followedBy2(start1, actRC)$
	$\mathcal{C}_{RC} \equiv Task \sqcap (\exists followedBy2.$ $\mathcal{C}_{CC}) \sqcap (\exists followedBy2.\sqcap$ $\mathcal{C}_{SPIC}) \sqcap = 2\, followedBy2$	$Task(actCC), SubProcess(spIC),$ $followedBy2(actRC, CC)$ $followedBy2(actRC, spIC)$

It is evident that the $MProcess2$ and $InstallComponent$ share different workflow relations $followedBy2$ and $followedBy4$ respectively. In [8] the authors do not mention any way of combining workflows for a main process and its constituent sub-processes. Our modeling framework is able to combine the control flow for the process and its sub-processes by using SWRL rules. For example, we can define a control flow relation $followedBy$ at the main process level and specify that the a task in Maintenance Process 2 is followed by the next task in sub-process 'Install Component' by use of the following rule:

$$followedBy(?x, ?z) \leftarrow Process(?p), Node(?x), SubProcess1(?sp), followedBy2(?x, ?sp),$$
$$beginsWith1(?sp, ?s), Start(?s), Node(?z), followedBy4(?s, ?z)$$

Contributions. In this work, we adopt the framework of hybrid programs which integrates OWL-DL ontology with SWRL rules, to model and specify business processes. We adhere to a decidable fragment of SWRL rules *a la* DL-safe rules [15] to make querying and other reasoning tasks decidable in this framework. Not only we are able to model the control flow of the process capturing hierarchical and ordering relationships between nodes for querying, we also advocate an explicit modular design of process ontology that helps building the ontology in stages. We also adequately model ABox axioms to maintain the consistency at each stage of the creation of ontology. This kind of consistent and modular ontology design can facilitate efficient process modeling in different industrial applications and business knowledge management.

The paper is organized as follows. In Sect. 2 we briefly describe the main notions of a business process. Process models are formalized using OWL-DL with SWRL rules in Sect. 3. We study patterns for querying and retrieving processes in Sect. 4. In Sect. 5 we describe an implementation of our framework and narrate our experimental effort. A case study of an industrial business process modeling in this formalism is described in Sect. 6. The discussion on related work is in Sect. 7. Finally we conclude in Sect. 8.

2 A Primer on Business Process

In this paper we work with Business Process Diagrams (BPD) captured using standard Business Process Modeling Notation (BPMN), which consist of nodes and control flow relation linking two nodes. A *node* can be a task (also called an activity here), an event, a fork (`AND-split`), a choice (`XOR-split`), a synchronizer (`AND-join`), and a merge (`XOR-join`) gateway. In a BPD, there are *start events* (also called *start nodes*) denoting the beginning of a process, and *end events* (also called *end nodes*) denoting the end of a process. A *start activity* is an activity which follows a start node. An activity is called a *sink activity* (or *end activity*) if it is immediately followed by an end event. We do not take into consideration message passing, timer events etc. in our model. A process can reside within another process. In this case, the former is called a *sub-process invocation activity* or simply *sub-process*. A business process is *well-formed* if it has exactly one start node with no incoming edges and one outgoing edge from it, there is only one incoming edge to a task and exactly one outgoing edge from a task, each fork and choice has exactly one incoming edge and at least two outgoing edges, each synchronizer and merge has at least two incoming edges and exactly one outgoing edge, every node is on a path from the start node to some end node, and there exists at least one task in between two gateways (this is to avoid triviality). Also we can safely assume that an end event is immediately preceded by a task, in absence of which we can introduce a dummy/silent task (which does not do anything). From now on without loss of generality, we shall consider only well-formed business processes.

The semantics of control elements of a business process are similar to that of work-flows discussed in [14]. There are two typical control flow related errors that can take place in processes: deadlock and lack of synchronization. A deadlock implies that the process will never terminate. A lack of synchronization allows multiple instances of the same task to occur in a process. A business process is *sound* if it does not produce deadlock and lack of synchronization. There are standard techniques to check the soundness of processes, for example see [4]. For this work, we consider only sound processes.

3 Formalizing Business Processes in OWL

We shall extract OWL-DL ontologies out of business processes in a constructive manner using OWL-DL [1,18] in conjunction with SWRL rules [16]. The satisfiability of OWL-DL in conjunction with SWRL is undecidable [15] as the latter is not DL-safe. It can be made decidable by adding a non-DL atom of the form $O(?x)$ in the body of a rule for every variable $?x$ appearing in it and adding a fact $O(a)$ for every individual a to the list of ABox axioms. In our case it would be possible to turn all the rules that we use into DL-safe rules.

3.1 Process Vocabulary

For extracting ontology out of arbitrary business processes we decompose the diagram into atomic patterns (originally introduced by Van der Aalst *et al.* for workflow diagrams in [20]) and generate an independent OWL concept for each of the patterns. Then using equivalence (and substitution) the concepts are merged to get the complex concept for the whole diagram in a modular fashion. The complex concept is generated in a top-down fashion from the model starting from the start activity of the diagram to an end activity. Our method is motivated by a construction of CSP[3] process from a business process [17] and conversion UML models to CSP expressions [2].

Below is the description of vocabulary for processes. There is a base class $Process$ representing the Business Process Diagram in question. $ConceptTask$ stands for a complex concept representing a pattern of a task, it will have sub-concepts $ConceptTaskA, ConceptTaskB$ corresponding to tasks A, B respectively and so on. Nodes are represented by the class $Node$. It has subclasses, $Task$ for a task, $Start$ for a start node, End for a final node and $\mathcal{J}$ for a synchronizer. The control flow relation is modeled by the role $followedBy$ which is defined on nodes. For composing different sub-process diagrams the role $followedBy$ for each diagrams may be denoted with a subscript (*e.g.*, $followedByi$), (where $followedByi \sqsubseteq followedBy$). Thus we may use $followedBy1$ for denoting the control flow for the sub-process at its first level embedding in the process, $followedBy0$ for the control flow relation for the main process etc. Finally the role $followedBy$ is defined to denote the control flow relation of whole process. The transitive role $followedByTran$ is a super role of $followedBy$. We use the role $contains$ with $Process$ as domain and $Node$ as range to denote that a process contains certain nodes. Further, there is a role $containedIn$ with $Node$ as domain and $Process$ as range to denote that a node is contained in a certain process. There is also a role $beginsWith$ which indicates that a process begins with a certain start node, similarly the role $endsWith$ says that a process ends with an end node. Table 2 describes the roles used for building the process ontology. Similar conventions using suffix like the role $followedBy$ for these roles, can be introduced when we deal with sub-processes. Lastly, we also consider a datatype role $pairWith$ on $Task$ and is string valued. This is used to capture the conditions on the outgoing edges of a XOR-split which can assume mutually exclusive values.

3.2 Modeling Basic Control Flow Patterns in OWL-DL

We consider basic control flow patterns [20] of a business process (see Fig. 3(a)), and translate them to OWL-DL complex concepts preserving the graphical structure of the process diagrams. This is done using a modular approach in which

[3] CSP here stands for Communicating Sequential Process.

Table 2. The roles used for building the process ontology (using ABox axioms)

Role	Type of properties	Description
$followedBy(node1, task2)$	Object	Node *node*1 is followed by task *task*2
$beginsWith(process1, start2)$	Object	The Business Process *process*1 starts with start node *start*2
$endsWith(process1, end3)$	Object	The Business Process *process*1 ends with end node *end*3
$contains(process1, task2)$	Object	The Business Process *process*1 contains a task node *task*2
$containedIn(task3, process2)$	Object	A task node *task*3 occurs in the Business Process *process*2
$pairsWith(task4,$ "*yes*")"	Datatype	Instantiated Task *task*4 is associated with the condition denoted by the string "*yes*" on an XOR-split outgoing edge

each pattern is modeled as an OWL concept (which can be thought of a module). These concepts are then suitably merged to obtain the final ontology. The atomic patterns used for process ontology are shown in Fig. 3(b). Below we supply the TBox axioms for each pattern. Later we shall provide corresponding ABox axioms.

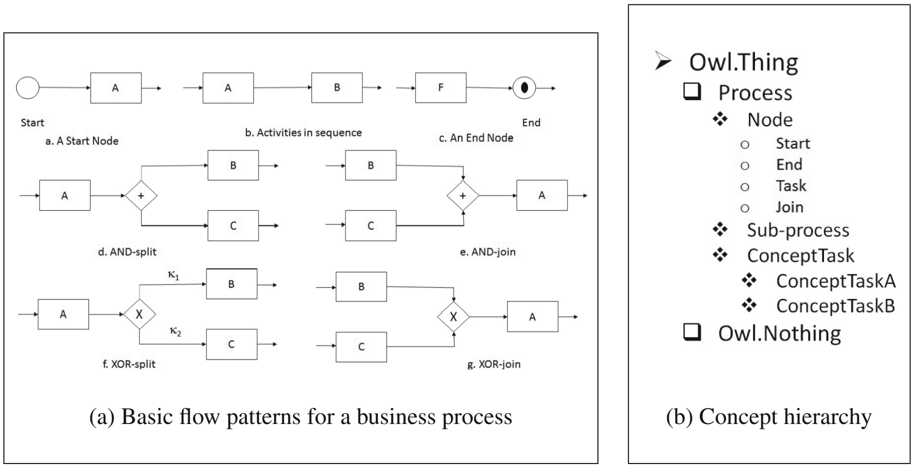
Start Event: As mentioned earlier, a *Start* activity is defined for the corresponding start event in the process. The expression $\mathcal{C}_0$ denotes the complex concept for the whole process (this is also called concept *Process*). The concept expression for the pattern in Fig. 3(a) is modeled as a conjunction of Start class and the existential restriction of concept class for activity A with $followedBy$ role. This convention will be followed throughout for generating concept expressions for other patterns of a business process.

$$\mathcal{C}_0 \equiv (Start \sqcap = 1 followedBy.\mathcal{C}_A).$$

Sequential Activities: In Fig. 3(a)(b) task A is followed by task B. Let $\mathcal{C}_A$ denote the concept associated with the task node A[4]. We assume that concept $\mathcal{C}_A$ is equivalent to the task class $Task$ and a following existentially quantified concept $\mathcal{C}_B$.

$$\mathcal{C}_A \equiv \ (Task \sqcap \ = 1 followedBy.\mathcal{C}_B).$$

[4] Denoted with a small rectangle.



(a) Basic flow patterns for a business process

- Owl.Thing
  - Process
    - Node
      - Start
      - End
      - Task
      - Join
    - Sub-process
    - ConceptTask
      - ConceptTaskA
      - ConceptTaskB
  - Owl.Nothing

(b) Concept hierarchy

Fig. 3. Pattern-based OWL ontology modeling for business processes

End Activity: In particular, for the pattern shown in Fig. 3(a)(c) the process terminates with an end node[5] which is preceded by a task F. We capture this as,

$$\mathcal{C}_F \equiv \ (Task \sqcap \ = 1 \ followedBy.End).$$

Parallel Split (AND-split): To ensure that the parallel concepts originating at an AND-split are synchronized (see Fig. 3(a)(d)) we use an axiom that states there exist multiple follower sequences, which are captured in terms of an intersection of sequences. In the conjunct we also use a cardinality constraint role to denote the number of outgoing edges out of the AND-split.

$$\mathcal{C}_A \equiv Task \ \sqcap \ (\exists followedBy.\mathcal{C}_B) \sqcap \ (\exists followedBy.\mathcal{C}_C) \ \sqcap \ (= 2.followedBy).$$

Synchronizer (AND-join): The behavioral property of the synchronizer states that it can be executed only after the end of the execution of all the input activities. For example, in Fig. 3(a)(e) A can be executed only after both B and C are executed. We define another class for AND-join (subclass of node $Node$), denoted as $\mathcal{J}$ (stands for $Join$ class).

$$\mathcal{C}_B \equiv \ Task \sqcap \ = 1.followedBy.(\mathcal{J} \sqcap \ = 1.followedBy.\mathcal{C}_A).$$
$$\mathcal{C}_C \equiv \ Task \sqcap \ = 1.followedBy.(\mathcal{J} \sqcap \ = 1.followedBy.\mathcal{C}_A).$$

[5] Denoted with a shaded circle inscribed within another circle.

Exclusive Choice (XOR-split): This operator becomes enabled when the input activity is executed and subsequently, one of the output activities is triggered (see Fig. 3(a)(f)). The conditions associated with the choice are captured on the destination nodes of the split gateways as string values for the data type property $pairsWith$. Below the symbol "$\cdots$" denotes other appropriate concepts appearing in the conjuncts for the concepts $\mathcal{C}_B$ and $\mathcal{C}_C$.

$$\begin{aligned}
\mathcal{C}_A &\equiv Task \ \sqcap \ (= 1 followedBy.(\mathcal{C}_B \sqcup \ \mathcal{C}_C)) \\
\mathcal{C}_B &\equiv Task \sqcap \exists pairsWith\text{“}xsd : string\text{”} \sqcap \cdots \\
\mathcal{C}_C &\equiv Task \sqcap \exists pairsWith\text{“}xsd : string\text{”} \sqcap \cdots
\end{aligned}$$

Exclusive Merge (XOR-join): If one of the incoming edges is executed, an XOR-merge is enabled leading to the execution of an appropriate activity on its outgoing edge, see Fig. 3(a)(g).

$$\begin{aligned}
\mathcal{C}_B &\equiv \ Task \sqcap \ = 1.followedBy.\mathcal{C}_A \\
\mathcal{C}_C &\equiv \ Task \sqcap \ = 1.followedBy.\mathcal{C}_A
\end{aligned}$$

Iteration of Activities. Each loop is assumed to contain a choice and a join, this is to avoid interminable looping. Thus a loop may be executed a certain number of times depending on the exit condition. Such a process can be specified by breaking it into patterns and writing the concepts accordingly. Consider an example of a loop in Fig. 4(a). Here the process begins with task A. Then tasks B and C can occur in a loop if the condition at the split gateway is false. If the condition evaluates to true then activity D occurs which marks the end of the process. Notice there is a choice gateway after B which splits into C and D, and there is a merge gateway which joins A and C to produce B. Then we write concepts corresponding to these gateways as formulated before. This may lead to cyclic terminology which need not be definitorial [1]. However, we can argue that this terminology will have an interpretation which is a fixpoint and hence will have a model. The cycle (due to the loop) in the dependency graph of this terminology contains zero negative arc and so, it will have a fix point interpretation [1].

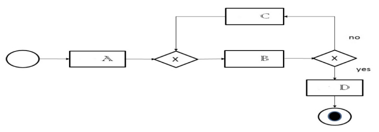
$$\begin{aligned}
\mathcal{C}_0 &\equiv (Start \sqcap = 1 followedBy.\mathcal{C}_A) \\
\mathcal{C}_A &\equiv (Task \sqcap \ = 1 followedBy.\mathcal{C}_B) \\
\mathcal{C}_B &\equiv Task \ \sqcap \ (= 1 followedBy.(\mathcal{C}_C \sqcup \mathcal{C}_D)) \\
\mathcal{C}_C &\equiv Task \sqcap \exists pairsWith\text{“}xsd : string\text{”} \sqcap = 1 followedBy.\mathcal{C}_B \\
\mathcal{C}_D &\equiv Task \sqcap \exists pairsWith\text{“}xsd : string\text{”} \sqcap = 1 followedBy.End
\end{aligned}$$

Sub-process Invocation Activities. A process may have a sub-process invocation activity which denotes another process. In Fig. 4(b), the original process P begins with the activity A which is followed by a sub-process invocation activity SP, which in turn, is followed by another activity B. Further in SP activity C is preceded by a start node and is followed by an end node. We write some of the axioms and rules which model this process. The last two rules link the control flow relation of the original process with that of the sub-process. These start and end activities of the main process and its sub-process will be disjoint from each other.

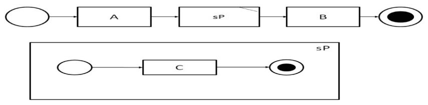
$$
\begin{aligned}
&Process \equiv (Start \sqcap =1 followedBy0.\mathcal{C}_A); SubProcess1 \equiv (Start \sqcap =1 followedBy1.\mathcal{C}_C)\\
&\mathcal{C}_A \equiv (Task \sqcap\ =1 followedBy0.\mathcal{C}_{SP}); \mathcal{C}_{SP} \equiv (SubProcess1 \sqcap\ =1 followedBy0.\mathcal{C}_B)\\
&\mathcal{C}_B \equiv\ (Task \sqcap\ =1\ followedBy0.End); \mathcal{C}_C \equiv\ (Task \sqcap\ =1\ followedBy1.End)\\
&SubProcess1 \sqsubseteq Process; followedBy0 \sqsubseteq followedBy; followedBy1 \sqsubseteq followedBy\\
&Process \sqsubseteq \exists followedBy0.Node.SubProcess1 \sqsubseteq \exists followedBy1.Node\\
&Process \sqsubseteq \exists contains0.Node.SubProcess1 \sqsubseteq \exists contains1.Node\\
&Process \sqsubseteq \exists beginsWith0.Start.SubProcess1 \sqsubseteq \exists beginsWith1.Start\\
&Process \sqsubseteq \exists endsWith0.End; SubProcess1 \sqsubseteq \exists endsWith1.End\\
&followedBy(?x, ?z) \leftarrow Process(?p), Node(?x), contains0(?p, ?x), SubProcess1(?sp),\\
&\ followedBy0(?x, ?sp), beginsWith1(?sp, ?s), Start(?s), Node(?z), contains1(?sp, ?z),\\
&\ followedBy1(?s, ?z)\\
&followedBy(?x, ?z) \leftarrow Process(?p), SubProcess1(?sp), Node(?x), contains1(?sp, ?x),\\
&\ endsWith1(?sp, ?e), End(?e), followedBy1(?x, ?e), Node(?z), contains0(?p, ?z),\\
&\ followedBy0(?sp, ?z).
\end{aligned}
$$

We add few more axioms in the process ontology some of which are borrowed from [8]; they are listed in Table 3. Here $\perp$ denotes contradiction, akin to owl:Nothing.

The above set of axioms constitute the TBox axioms for the ontology created corresponding to any business process. The process diagram itself serves as the meta-model for the process from which those (TBox) axioms are to be extracted, whereas the specific process model would lead to the generation of ABox axioms. We consider the process patterns in Fig. 3(a) and list sample ABox axioms corresponding to those basic patterns in Table 4. In this way we can construct the ontology for a business process which can be shown to be consistent by taking induction on the structure of the process and showing that ontology (TBox and ABox axioms) corresponding to each pattern is consistent, a formal argument of which is left out for a future work.



(a) An Example of a Loop



(b) A Process with sub-process invocation activity

Fig. 4. Other patterns for business processes

Table 3. Axiomatization for business process diagrams

Statement	OWL-DL axioms
Start and End activities and Task are subclasses of class node	$Start, End, Task \sqsubseteq Node$
An AND-join is a subclass of class node	$Join \sqsubseteq Node$
$followedByTran$ is the super role of $followedBy$	$followedByi \sqsubseteq followedBy$, and $followedBy \sqsubseteq followedByTran$ $(i = 0, 1, \ldots)$
Start Activity has no predecessor	$(Node \sqcap \ \forall containedIn.Process \sqcap \ = 1 followedBy.Start) \sqsubseteq \bot$
End Activity has no follower	$(End \sqcap \ = 1 followedBy.Node) \sqsubseteq \bot$
A Process contains some nodes	$Process \sqsubseteq \exists contains.Node$
A Process begins with a start node	$Process \sqsubseteq = 1 beginsWith.Start$
A Process ends on an end node	$Process \sqsubseteq \exists endsWith.End$
$contains$ and $containedIn$ shares a reciprocal relation	$containedIn(?y, ?x) \leftarrow contains(?x, ?y)$ $contains(?x, ?y) \leftarrow containedIn(?y, ?x)$

Table 4. ABox axioms for basic patterns

Basic patterns	ABox axioms
Start event	$Process(process1), Start(startS), Task(taskA), followedBy(startS, taskA),$ $contains(process1, startS), containedIn(startS, process1)$ etc
Sequential activities	$Task(taskA), Task(taskB), followedBy(taskA, taskB)$ etc.
End event	$End(endS), Task(taskF), followedBy(taskF, endS)$ etc.
Parallel split	$Task(taskA), Task(taskB), Task(taskC),$ $followedBy(taskA, taskB), followedBy(taskA, taskC)$
Synchronizer	$Task(taskA), Task(taskB), Task(taskC), Join(join1),\ followedBy(taskB, taskA),$ $followedBy(taskC, taskA),\ followedBy(taskB, join1), followedBy(taskC, join1)$
Exclusive choice	$Task(taskA), Task(taskB), Task(taskC),\ followedBy(taskA, taskB),$ $followedBy(taskA, taskB),\ pairsWith(taskB, \text{“}yes''), pairsWith(taskC, \text{“}no'')$
Exclusive merge	$Task(taskA), Task(taskB), Task(taskC),$ $followedBy(taskB, taskA), followedBy(taskC, taskA)$

4 Semantic Query Patterns for Querying and Retrieving Processes

We now specify a few requirements on the process models shown in Fig. 1 to query them and retrieve process information. These queries will be posed using DL query on three processes, $MProcess1$ (Maintenance Process 1), $MProcess2$ (Maintenance Process 2) and $MProcess3$ (Maintenance Process 3).

4.1 Querying Processes

We show how we capture requirements for an individual process using the proposed ontology modeling. In particular, we consider Maintenance Process 1. Below **Req** will stand for requirement and **Spec** will denote the corresponding specification of the requirement.

- **Req1** Cleaning component is preceded by removing component.
  Spec1: $Start \sqcap (\exists followedByTran(value``actRC'' \sqcap \exists followedByTran\ value``actCC''))$.
- **Req2** Repairing component is always followed by installing the component.
  Spec2: $Start \sqcap \exists followedByTran(value``actRep'' \sqcap \exists followedByTran\ value ``actIC'')$
  $\sqcap \neg \exists followedByTran(value``actRep'' \sqcap \exists followedByTran\ value``actX''$
  $\sqcap differentFrom(actX, actIC))$.
- **Req3** Installing component can mark the end to the maintenance process.
  Spec3: $Start \sqcap \exists followedByTran(value``actIC'' \sqcap \exists followedBy.End)$.

4.2 Querying for Process Retrieval

We use three different query patterns with respect to the control flow of the process [8] to retrieve processes. The first pattern is about the execution order. The second pattern is used to query a process for modality. The third pattern is related to terminology which facilitates merging flow relations from the original process and its sub-process. The **Query** input depicts a process description. **Result** prints out the relevant processes (as individuals). We use DL queries for querying using concept subsumption as inference.

Pattern for Execution Order. A query related to execution pattern is described below.

Query: Which processes do execute Inspect Component after Remove Component?
$\{P \equiv Start \sqcap \exists followedByTran.(value``actRC'' \sqcap \exists followedByTran.value ``actIC'')\}$
Result: {MProcess1, MProcess2, MProcess3}
The result contains all processes that execute Inspect Component after Remove Component with an arbitrary number of activities between them.

Pattern for Process Modality. This pattern deals with queries which refers to the modality of the processes as described below.

Query: Which process does offer Replace Component?
$\{P \equiv \exists contains.\, value\text{“}actRC\text{”}\}$

Result: {MProcess3}
The query will search for the processes that are related to Replace Component via the *contains* role. Only the MProcess3 has the Replace Component Task.

Pattern for Process Terminology. By using this pattern one can query details about the original process and its constituent sub-process using terminological knowledge.

Query: Which processes execute Declare fitness before Install Component?
$\{P \equiv Start \sqcap \exists followedByTran.(value\text{“}actDF\text{”} \sqcap \exists followedByTran.value\text{“}actIC\text{”})\}$
Result: {MProcess1, MProcess2, MProcess3}
As Declare Fitness is related to Install Component using the merged control flow relation of the original process and the sub-process (captured using SWRL rules), we can query processes with different levels of process hierarchy.
Query: Which process does allow the execution of Clean Component and Test Component in an interleaving manner?
$\{P \equiv Start \sqcap ((\exists followedByTran.(value\text{“}actCC\text{”} \sqcap \exists followedByTran.value\text{“}actTC\text{”})$
$\sqcup(\exists followedByTran.(value\text{“}actTC\text{”} \sqcap \exists followedByTran.value\text{“}actCC\text{”})\}$
Result: {MProcess2}
The process obtained thus allows an interleaving of the execution of Clean Component and Test Component.

5 Experimental Results

In this section we describe our implementation efforts. Also we report on the experiments that we perform on industrial process models.

5.1 Implementation

We implement our framework using Java and OWL API. At the front end we use a BPM modeler for capturing business process models. For this purpose we use Camunda[6] which is an open source platform for workflow and business process management. Camunda Modeler provides a desktop application for editing BPMN process diagrams (using BPMN 2.0) which comes with a graphical user interface. Moreover, a business process drawn in Camunda can be exported as an xml file. We have written a translator in Java that can parse the xml

[6] Available at https://camunda.org/.

file, generate basic patterns out of the process and create OWL ontology with SWRL rules for each of them using OWL API which are subsequently merged. The OWL ontology can be uploaded on Protégé framework for viewing. A Java implementation with some example processes can be found in "https://github.com/gsilvatici/DiagramOntologyParser".

5.2 Experiments and Evaluation

Dataset. For experimental purposes we consider a total of 29 different industrial process models. These processes are modeled on Infosys in-house requirements modeling tool called InFlux Requirements Studio (RS).[7]. Influx RS has a process modeling editor interface which uses BPMN for capturing processes and produces xmi (a kind of xml) output files for these processes. We consider these xmi files produced from processes modeled on InFlux and translate them into OWL using our algorithm. These process models contain tasks, exclusive gateway, parallel gateways, loops and sub-processes. We create a KB for evaluating the process retrieval by using process multipliers on original processes.

Methodology. For the evaluation we use the HemriT 1.3.8.413 reasoner in Java 1.8 running on a computer with 3.2, 4-core GHz CPU and 16 GB RAM. We create four knowledge bases with different sizes (with processes chosen with replacement) on which process retrieval is evaluated. The first KB contains 9, the second 15, and the third 20 and fourth contains 29 process models. The Protègè 5.2 framework (with HemriT 1.3.8.413) is used for querying and reasoning. For each of the KBs, the evaluation consists of 10 to 20 queries similar to the queries illustrated in Sect. 4.

Result. We present our evaluation result concerning process retrieval in Table 5. KB size refers to the number of processes and their constituent sub-processes. By KB node size we mean the number of nodes (activities, gateways and events) of the processes comprising the KB. The quantity sub-process per process denotes the (average and maximum) number of sub-processes in each process in the KB. We also capture the number of axioms and rules present in each KB. We show the (average and maximum) retrieval time (in milliseconds (ms)) for both simple and complex queries. Simple queries include only one activity or a negated activity. Complex queries consist of multiple (at least three) activities. The average retrieval time for concept satisfiability is also shown. By concept satisfiability[8] we mean checking the consistency of the conjunction of the complex concepts for a process with all other relevant axioms like class hierarchy, object property hierarchy, datatype property hierarchy, class assertions, object property assertions and the like.

[7] InFlux process models have created high impact on Infosys business by bringing in formalism and repeatability into the process of translation of business objectives into IT solutions.

[8] This is like consistency checking of the whole ontology created on Protégé.

Table 5. Result on process retrieval

No.	KB size	KB node size	Sub-process per process size		KB axioms	KB rules	Simple query time [msec]		Complex query time [msec]		Concept sat time [msec]
			Avg	Max			*Avg*	*Max*	*Avg*	*Max*	*Avg*
1	20	151	2	3	1746	179	4390	4482	4201	4270	6806
2	20	143	3	5	1632	183	5820	6020	5750	5826	18909
3	29	214	2	3	2498	258	6721	6851	6515	6684	20310
4	29	217	3	5	5442	261	12213	12773	12228	12307	42322

The performance evaluation has the following outcomes. The number of sub-processes in a process makes considerable difference in query retrieval and concept satisfiability time, as evidenced by the 3rd and 4th KB. Both these KBs have the same process size of 29, but the average number of sub-processes per process in 3rd KB is higher than that in 4th KB, so much so, the retrieval time of queries differs by almost 100% for them. The 1st and the 2nd KB show comparable performances. However, the 1st KB though having larger KB node size, shows lower retrieval time for simple and as well as complex queries. It seems the retrieval time is also greatly influenced by the number of sub-processes per process in the KB as mentioned before. This is because the number of rules in the KB increases with the increase in the number of sub-processes (every sub-process generates two lengthy SWRL rules). It is interesting to note that except for the 2nd KB, the retrieval time for both kinds of queries goes up with an increase in KB node size.

The retrieval time for queries is directly proportional to KB size. As indicated for all the KBs, the retrieval time for queries is more than 50% high for a KB with 29 processes than the KB with 20 processes, while the number of sub-process per process remains almost constant. Also the number of sub-processes per process seems to affect the concept satisfiability time a lot. For the same KB size of 1st and 2nd KBs, concept satisfiability time increases by three times for the latter. Similar trend is shown in 3rd and 4th KBs, the concept satisfiability time for 4th KB is two times more than 3rd KB.

The evaluation realizes the following qualitative results. (1) Processes from the KB are subsumed by the more general query processes using the roles $followedBy$ and $followedByTran$. (2) The presence of unique activities inside a process lends an advantage on the use of the *contains* role, as using it with along the $followedBy$ role leads to a more expressive query. (3) The retrieval of processes with queries that use activities occurring inside sub-processes are made possible by the use of SWRL rules that links the control flow relation of the original process with that of its sub-processes at different levels of hierarchy.

Limitations. Our experimentation is restricted in scope due to KB size. This is because, for KB with more than 40 process models, concept satisfiability for most of the cases does not terminate. This is mainly due to the number of rules that are present in the KB (and of course, consistency checking and querying in OWL-DL is NEXPTime complete [15]). This is confirmed by the fact that consistency checking without SWRL rules in the KB leads to appreciable decrease in time. It is obvious that use of SWRL degrades the performance of querying and consistency checking. However, from Fig. 5 we can see that consistency (concept satisfiability) checking time is only about 45000 ms more for KB with no rules compared to KB with rules in limit. Also the former rises appreciably in comparison to the latter when the KB size is close to 20.

6 Business Process Modeling as Part of Knowledge Management: A Case Study

As the business centers are becoming more process-oriented they have started to adopt process modeling methods and tools to effectively manage the complexity of these systems. This has led to the development of integrated business process modeling tools which are capable of delivering valuable business objectives. Each BPM software application is made of a combination of several components: process modeling and design, process monitoring, process operation (automation and integration) and technology platforms and interfaces. Although each of these components is important for selecting a BPM tool the ability to model process and maintain process repository would be the most important yardstick for choosing such tools. Even if modeling tools adequately support the modeling and enactment of business processes, they still do not provide much support for knowledge-related activities. By proper ontology modeling of processes it will be possible to extract existing knowledge that can be made explicit to the user. This is the reason we consider a case study of a business process and follow a step-by-step approach of extracting OWL+SWRL ontology that explicitly integrates knowledge management activities into the business process environment.

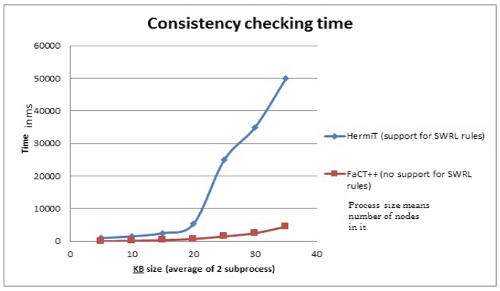


Fig. 5. Consistency checking time for KBs with and without rules

Figure 6 represents an example of business process in a banking domain (related to opening of a savings account) arising in an IT application scenario. This is an abridged version of a process model in Infosys business process repository. In this process the bank receives request (A) for opening an account from a client. The bank then obtains client information (B) and selects service (C). Getting client information reduces to getting driver's license (D) and analyzing customer relationship (E) whereas select service forks into record information (F) and submit deposit (G). After analyzing customer relation and recording information, document is prepared (H) followed by preparing for account opening (I). For final phase of account opening, a review process is scheduled (J)

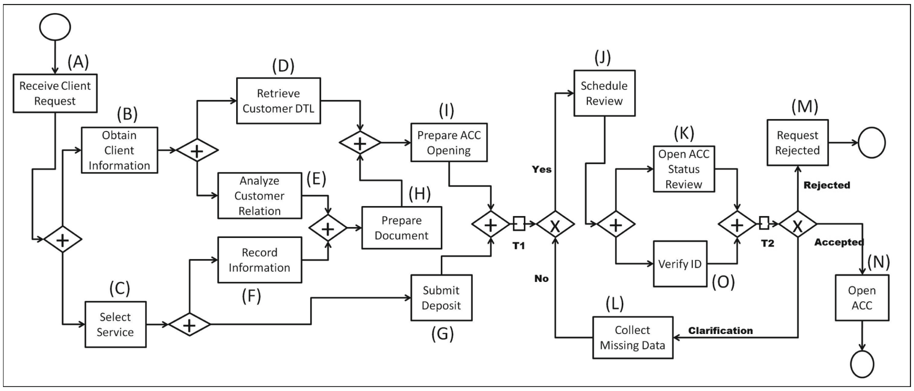


Fig. 6. An example of a bank account opening process

which follows account status review (K) and ID verification (O). Finally there is a decision where either a request is rejected (M), or an account is opened successfully (N), or missing data is sought (L) for. Note that the activities are marked using capital letters in parentheses corresponding to the activities in the diagram. We shall divide this process modeling effort into the following steps, - sub-process identification, vocabulary fixing, concept hierarchy design, pattern identification, pattern to ontology creation, axiom listing and ontology merging.

Sub-process identification. One needs to identify all the sub-processes as each sub-process will be characterized by a separate control flow relation. So, if there are n processes then we should have assigned n control flow relations, $followedBy1, \ldots, followedByn$. If a sub-process is embedded in another sub-process then we use another subscript, for example, if a sub-process with control flow relation $followedBy2$ contains another sub-process then the control flow of the latter will be denoted by $followedBy21$, and so forth. However, this particular process does not contain any sub-process, and hence can be modeled by a single control flow relation $followedBy$ and its transitive closure.

Vocabulary fixing. Once the control flow relation associated with each sub-process and the main control flow relation with the original process are decided one needs to identify a naming convention for all the nodes in the process. Activities are provided with named individuals, for example, task A will be represented as an individual $actA$, task B as $actB$ etc. Similarly, start and end nodes are instantiated as $start1, end1, \ldots$ etc. Each join node is also associated with an individual like join1, ... etc. The naming convention is that join nodes are marked from left to right in the figure with the word 'join' appended with appropriate natural numbers. The naming convention for roles will be as described in Sect. 3.1. For ease of notation we may adopt the unique name assumption which states that different names always refer to different entities in the world, although it is not a requirement in OWL modeling.

Concept hierarchy design. In most of the cases the concept hierarchy in Fig. 3(b) is adopted.

Pattern identification. It is known that a well-formed process can be decomposed into atomic patterns as proposed by Van der Aalst *et.al* in [20]. Hence we can identify those patterns (as depicted in Fig. 3(a)) in the process considered.

Pattern to ontology creation. Once a business process is associated with well-designated patterns one can start writing ontology for each of these patterns. For some of these patterns we specify TBox axioms as below. Also for these patterns we list the ABox axioms in Table 6.

Axiom listing. Some extra axioms like the ones in Table 3 are also added. For sub-processes similar axioms need to be listed specific to each of them.

Ontology merging. Finally, the ontology specified in the last two steps are merged. By construction the created ontology as argued in Sect. 3.2, will be consistent.

Table 6. A partial specification of ontology for the business process in Fig. 6

Patterns	TBox axioms	ABox axioms
Start node followed by task A	$\mathcal{C}_0 \equiv (Start \sqcap = 1 followedBy.\mathcal{C}_A)$	$Start(start1), Task(actA), followedBy(start1, actA)$
Task A is split into two parallel branches containing tasks B and C respectively	$\mathcal{C}_A \equiv Task \sqcap (\exists followedBy.\mathcal{C}_B) \sqcap (\exists followedBy.\mathcal{C}_C) \sqcap (= 2.followedBy)$	$Task(actB), Task(actC), followedBy(actA, actB), followedBy(actA, actC)$
Task B is split into two parallel branches containing tasks D and E respectively	$\mathcal{C}_B \equiv Task \sqcap (\exists followedBy.\mathcal{C}_D) \sqcap (\exists followedBy.\mathcal{C}_E) \sqcap (= 2.followedBy)$	$Task(actD), Task(actE), followedBy(actB, actD), followedBy(actB, actE)$
...	...	...
Tasks E and F get synchronized into task H	$\mathcal{C}_E \equiv Task \sqcap = 1.followedBy.(\mathcal{J} \sqcap = 1.followedBy.\mathcal{C}_H)$, $\mathcal{C}_F \equiv Task \sqcap = 1.followedBy.(\mathcal{J} \sqcap = 1.followedBy.\mathcal{C}_H)$	$Task(actH), Join(join1), followedBy(actE, actH), followedBy(actF, actH), followedBy(actE, join1), followedBy(actF, join1)$
...	...	...
Task N is followed by end node	$\mathcal{C}_N \equiv (Task \sqcap = 1\ followedBy.End)$	$Task(actN), End(end1), followedBy(actN, end1)$

In this way, one can create ontology for each process in a repository and merge those ontologies to create useful organizational knowledge repository. As nowadays, most of the organizations are involved in the projects related to business process management this work could facilitate adopting an integrated business process and knowledge management centric approach. Such a modeling framework should encourage the development of integrated BPM and Knowledge management software tools that should enable the transformation of business process models into knowledge repository.

7 Related Work

There are some existing research work on annotating business processes using semantic web formalisms. Business processes have been tagged with semantic labels as a part of knowledge base with a view to formalize business process structure, business domains, and a set of criteria describing correct semantic marking in [5]. In another work [6], the authors propose semantic web language OWL to formalize business process diagrams, and automatically verify sets of constraints on them, that deal with knowledge about the domain and process structure. There have been earlier attempts to model processes and retrieve process structures. Sequences arising in a business process have been modeled in [10]. Processes are retrieved using process information and annotation in [23]. Process reuse has been advocated in [7] by using DL-based process models, but complex control flow is not captured. In [9], the authors build on these approaches of process modeling, and attempt to analyze requirements for modeling and querying process models. They also present a pattern-oriented approach for process modeling and information retrieval. For this purpose they specify the execution order of a process in OWL and also express the modality and process activities and structures in the same formalism. In a recent work [11] Käfer and Harth design an ontology for representing workflows over components with Read-Write Linked Data interfaces and give an operational semantics to the ontology via a rule language, however they do not deal with explicit business process models. As mentioned before, our work offers some advantage over this work as it allows a modular design of process models and combines control flows of an original process and constituent sub-processes using SWRL rules.

These semantic annotation techniques of business processes lead to the possibility of semantic validation, *i.e.,* whether the tasks in business processes are consistent with respect to each other in the underlying framework of semantic annotation. Such issues are investigated in [22], where the authors introduce a formalism for business processes, which combines concepts from work flow with that from AI. A rule-based method for modeling processes and workflows has been proposed in [13], where the authors introduce an extended Event-Condition-Action (ECA) notation for refining business rules, and perform a consistent decomposition of business processes. Cicekli and Cickeli have used event calculus, a kind of logic programming formalism for specification and execution of workflows in [3]. They express control flow graph of a work flow specification as a set of logical formulas. Using a similar framework of Constraint Logic Programming the authors propose a method for representing and reasoning about business processes from both the workflow and data perspective [19]. However most of these techniques lead to undecidable reasoning problem.

Let us also contrast our OWL modeling of business processes with possible modeling of the same with OWL-S [21]. OWL-S is one of the first attempts to present an OWL ontology for semantic modeling of processes. OWL-S differentiates between atomic and composite processes. Our modeling technique does not require such a differentiation. OWL-S Sequence allows arbitrary processes as components. As a result, the semantics of a sequence of two splits is unclear and unintuitive. Suppose a split A has components $A_1, \ldots, A_m$ and split B has components $B_1, \ldots, B_n$. When A and B are composed in a sequence such that A is followed by B, it may still happen that a B_i is finished before an A_j. Our modeling technique requires the first component to be a start event. OWL-S process model proposes a complicated way for modeling input and output parameters with OWL. We do not model input and output parameters of activities in our approach.

8 Conclusion

In this work we provide a formalization of business processes in OWL-DL with SWRL rules for modeling and retrieving purposes. Our modeling technique offers some advantages over the existing repertoire by the way of offering a modular approach of modeling, linking control flow relations of a process and its subprocess using rules and providing a consistent way of modeling ontologies. The framework can be successfully integrated with different applications involving requirement engineering, business process information systems etc. In future we would like to design a natural language query interface for querying business processes in this framework. Sentences expressed in a controlled subset of natural language will be used as query languages on the user interface which will be converted to SQWRL queries at the back end for information retrieval.

References

1. Baader, F., Nutt, W.: Basic description logics. In: Description Logic Handbook, pp. 43–95 (2003)
2. Bisztray, D., Heckel, R.: Rule-level verification of business process transformations using CSP. ECEASST **6** (2007)
3. Cicekli, N.K., Cicekli, I.: Formalizing the specification and execution of workflows using the event calculus. Inf. Sci. **176**(15), 2227–2267 (2006)
4. Fahland, D., et al.: Analysis on demand: instantaneous soundness checking of industrial business process models. Data Knowl. Eng. **70**(5), 448–466 (2011)
5. Di Francescomarino, C., Ghidini, C., Rospocher, M., Serafini, L., Tonella, P.: Reasoning on semantically annotated processes. In: Bouguettaya, A., Krueger, I., Margaria, T. (eds.) ICSOC 2008. LNCS, vol. 5364, pp. 132–146. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-89652-4_13
6. Di Francescomarino, C., Ghidini, C., Rospocher, M., Serafini, L., Tonella, P.: Semantically-aided business process modeling. In: Bernstein, A., et al. (eds.) ISWC 2009. LNCS, vol. 5823, pp. 114–129. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-04930-9_8
7. Goderis, A., Sattler, U., Goble, C.A.: Applying description logics for workflow reuse and repurposing. In: Proceedings of the International Workshop on Description Logics (DL 2005) (2005)
8. Gröner, G., Staab, S.: Modeling and query pattern for process retrieval in OWL. In: Proceedings of the 5th International Conference on Knowledge Capture (K-CAP 2009), 1–4 September 2009, Redondo Beach, California, USA, pp. 189–190. ACM (2009)

9. Groener, G., Staab, S.: Modeling and query patterns for process retrieval in OWL. In: Bernstein, A., Karger, D.R., Heath, T., Feigenbaum, L., Maynard, D., Motta, E., Thirunarayan, K. (eds.) ISWC 2009. LNCS, vol. 5823, pp. 243–259. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-04930-9_16
10. Hirsh, H., Kudenko, D.: Representing sequences in description logics. In: Proceedings of the Fourteenth National Conference on Artificial Intelligence and Ninth Innovative Applications of Artificial Intelligence Conference, AAAI 1997, pp. 384–389 (1997)
11. Käfer, T., Harth, A.: Specifying, monitoring, and executing workflows in linked data environments. CoRR abs/1804.05044 (2018). http://arxiv.org/abs/1804.05044
12. Kiefer, C., Bernstein, A., Lee, H.J., Klein, M., Stocker, M.: Semantic process retrieval with iSPARQL. In: Franconi, E., Kifer, M., May, W. (eds.) ESWC 2007. LNCS, vol. 4519, pp. 609–623. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-72667-8_43
13. Knolmayer, G., Endl, R., Pfahrer, M.: Modeling processes and workflows by business rules. In: van der Aalst, W., Desel, J., Oberweis, A. (eds.) Business Process Management. LNCS, vol. 1806, pp. 16–29. Springer, Heidelberg (2000). https://doi.org/10.1007/3-540-45594-9_2
14. Liu, R., Kumar, A.: An analysis and taxonomy of unstructured workflows. In: van der Aalst, W.M.P., Benatallah, B., Casati, F., Curbera, F. (eds.) BPM 2005. LNCS, vol. 3649, pp. 268–284. Springer, Heidelberg (2005). https://doi.org/10.1007/11538394_18
15. Motik, B., Sattler, U., Studer, R.: Query answering for OWL-DL with rules. J. Web Semant. **3**, 41–60 (2005)
16. O'Connor, M., Tu, S., Nyulas, C., Das, A., Musen, M.: Querying the semantic web with SWRL. In: Paschke, A., Biletskiy, Y. (eds.) RuleML 2007. LNCS, vol. 4824, pp. 155–159. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-75975-1_13
17. Roy, S., Bihary, S., Laos, J.A.C.: A CSP-theoretic framework of checking conformance of business processes. In: 19th Asia-Pacific Software Engineering Conference, APSEC 2012, pp. 30–39 (2012)
18. Schmidt-Schauß, M., Smolka, G.: Attributive concept descriptions with complements. Artif. Intell. **48**(1), 1–26 (1991)
19. Smith, F., Proietti, M.: Reasoning on data-aware business processes with constraint logic. In: Proceedings of the 4th International Symposium on Data-Driven Process Discovery and Analysis (SIMPDA 2014), pp. 60–75 (2014)
20. Van Der Aalst, W.M.P., Ter Hofstede, A.H.M., Kiepuszewski, B., Barros, A.P.: Workflow patterns. Distrib. Parallel Databases **14**(1), 5–51 (2003)
21. W3C Recommendation: OWL-S Semantic Markup for Web Services (2004). http://www.w3.org/Submissions/OWL-S
22. Weber, I., Hoffman, J., Mendling, J.: Beyond soundness: on the verification of semantic business process models. Distrib. Parallel Databases **27**, 271–343 (2010)
23. Wolverton, M., Martin, D.L., Harrison, I.W., Thoméré, J.: A process catalog for workflow generation. In: Proceedings of 7th International Semantic Web Conference, ISWC 2008, pp. 833–846 (2008)