Ensuring the Semantic Correctness of Workflow Processes: An Ontological Approach

Thi Hoa Hue Nguyen, Nhan Le Thanh

Ensuring the Semantic Correctness of Workflow Processes: An Ontological Approach

Thi-Hoa-Hue Nguyen and Nhan Le-Thanh

WIMMICS - The I3S Laboratory - CNRS - INRIA
University of Nice Sophia Antipolis
Sophia Antipolis, France
nguyenth@i3s.unice.fr,Nhan.LE-THANH@unice.fr

Abstract. Workflow verification has been known as an important aspect of workflow management systems. Many existing approaches concentrate on ensuring the correctness of workflow processes at the syntactic level. However, these approaches are not sufficient to detect errors at the semantic level. This paper contributes to ensure the semantic correctness of workflow processes. First, we propose a formal definition of semantic constraints and an $O(n^3)$-time algorithm for detecting redundant and conflicting constraints. Second, by relying on the CPN Ontology (a representation of Coloured Petri Nets with OWL DL ontology) and sets of semantic constraints, workflow processes are semantically created. And third, we show how to check the semantic correctness of workflow processes with the SPARQL query language.

Keywords: Ensuring, Ontology, Semantic Constraint, Semantic Correctness, SPARQL query, Workflow Process

1 Introduction

Workflows have drawn an enormous amount of attention from the research community and the industry. Over the years, workflow technology has been applied extensively to the business area. So far, various researchers have focused on process specification techniques [1], [2] and on conceptual models of workflow [3], [4]. However, specifying a real-world business process is mostly manual and is thus prone to human error, resulting in a considerable number of failed projects. Therefore, model quality, correctness and re-usability become very important issues. Although numerous approaches have been already developed to ensure workflow correctness at the syntactic level (e.g., avoiding deadlocks, infinite cycles, etc.). At the semantic level there may exist errors. Recently, few researches focus on checking the semantic conformance of workflow processes.

To check the semantic correctness of a model, we consider semantic constraints as *domain specific restrictions on a business process which need to be conformed during the process is executed*. Let us take an example: In a process for the Order Management activity, when an order is accepted, an order confirmation must be sent out later; If no order confirmation is scheduled to be sent, or this activity is not done in the right position, a semantic error occurs.

The work in [5] provides a very useful inspiration for our work, but it does not discuss how to formulate semantic constraints and also does not mention about the control-flow perspective in process models. Our objective is to support workflow designers in generating semantically rich workflow processes which allow syntactic and semantic verification. In this paper, we restrict ourselves to ensure the semantic correctness of workflow processes. Our contributions are:

- Giving a formal method to describe a variety of semantic constraints;
- Proposing an algorithm to check redundant and conflicting semantic constraints;
- Developing an ontology for annotating semantic constraints and representing control flow-based business workflow processes based on that ontology;
- Showing how to use the SPARQL query language [6] to check the semantic correctness of workflow processes.

This paper is organized as follows: in Section 2, we discuss related work. A short introduction to the CPN Ontology, which is defined to represent Coloured Petri Nets (CPNs) with OWL DL, is given in Section 3. Section 4 proposes a formal definition of semantic constraints for business processes. An algorithm used for checking redundant and conflicting semantic constraints is presented. We then develop a semantic conformance-oriented ontology. In Section 5, we present the creation of correspondences between those two ontologies to develop workflow processes. These workflows enable designers to check their semantic correctness. Five semantic verification issues of a workflow process are introduced in Section 6. Finally, Section 7 concludes the paper with an outlook on the future research.

2 Related Work

Today, software systems that automate business processes have become more and more advanced. Various researchers have paid attention to the problem of ensuring the correctness of process models. Many methods have been done in workflow verification. Most of them focus on the control-flow perspective, such as [7], [8], [9], [10], [11] to prevent errors (e.g., avoiding deadlocks, infinite cycles) at the syntactic level. Nevertheless, they check mainly the conformance of a workflow process based on the principle that if the constraints on data and control flow are met during execution, the workflow is correct.

Recently, a number of researches have grown beyond the scope of pure control-flow verification. Some approaches analyze the final state of the whole workflow [12] or consider running processes [13], [14], [15]. In [13], semantic constraints are defined over processes which are used to detect semantic conflicts caused by violation only of dependency and mutual exclusion constraints. They presented techniques to ensure semantic correctness for single and concurrent changes at process instance level. With regard to ontological approaches, aspects of semantic correctness are considered in few researches, such as [5], [16], [17]. The approach of [17] requires both the annotation of preconditions and effects to

ensure the models are semantically correct. In [5], the SPARQL query language is used to check the semantic correctness of ontology-based process representations. Individual model elements were annotated with concepts of a formal ontology, and constraints were formalized as SPARQL queries. However, these approaches depend on the availability of an ontology and a bulky annotation of process models.

We know that the ontology-based approach for modelling business process is not a new idea. There are some works made efforts to build business workflow ontologies, such as [18], [4], [19] to support (semi-)automatic system collaboration, provide machine-readable definitions of concepts and interpretable format. However, the issue that relates to workflow verification at the syntactic level is not mentioned. By extending the state-of-the-art, we use the Web Ontology Language to define the CPN Ontology for representing CPNs with OWL DL ontology and the semantic conformance-oriented ontology. Our ontological approach enables to create high quality and semantically rich workflow processes.

3 Representation of Coloured Petri Nets with OWL DL ontology

In this Section, we introduce the CPN Ontology [20] defined for business processes modelled with CPNs. The purpose is to ensure the syntactic correctness of workflow processes and to facilitate business process models easy to be shared and reused.

On one hand, Coloured Petri Nets (CPNs) [21] have been developed into a full-fledged language for the design, specification, simulation, validation and implementation of large software systems. Consequently, modelling business processes with CPNs supports workflow designers easy to verify the syntactic correctness of workflow processes [8]. On the other hand, **OWL DL** [22], which stands for OWL Description Logic, is equivalent to Description Logic $\mathcal{SHOIN}(\mathcal{D})$. OWL DL supports all OWL language constructs with restrictions (e.g., type separation) and provides maximum expressiveness while keeping always computational completeness and decidability. Therefore, we choose OWL DL language to represent the CPN Ontology. We believe that the combination of CPNs and OWL DL provides not only semantically rich business process definitions but also machine-processable ones. Figure 1 depicts the core concepts of the CPN ontology.

The CPN Ontology comprises the concepts: **CPNOnt** defined for all possible CPNs; **Place** defined for all places; **Transition** defined for all transitions; **InputArc** defined for all directed arcs from places to transitions; **OutputArc** defined for all directed arcs from transitions to places; **Token** defined for all tokens inside places (We consider the case of one place containing no more than one token at one time); **GuardFunction** defined for all transition expressions; **CtrlNode** defined for occurrence condition in control nodes; **ActNode** defined for occurrence activity in activity nodes, **Delete** and **Insert** defined for all expressions in input arcs and output arcs, respectively; **Attribute** defined for all

$$
\begin{aligned}
&CPNOnt \equiv \geq 1 hasTrans.Transition \sqcap \geq 1 hasPlace.Place \\
&\quad \sqcap \geq 1 hasArc.(InputArc \sqcup OutputArc) \\
&Place \equiv connectsTrans.Transition \sqcap \leq 1\ hasMarking.Token \\
&Transition \equiv connectsPlace.Place \sqcap = 1 hasGuardFunction.GuardFunction \\
&InputArc \equiv \geq 1 hasExpresion.Delete \sqcap \exists hasPlace.Place \\
&OutputArc \equiv \geq 1 hasExpression.Insert \sqcap \exists hasTrans.Transition \\
&Delete \equiv \forall hasAttribute.Attribute \\
&Insert \equiv \exists hasAttribute.Attribute \\
&GuardFunction \equiv \geq 1 hasAttribute.Attribute \sqcap = 1 hasActivity.ActNode \\
&\quad \sqcup = 1 hasControl.CtrlNode \\
&Token \equiv \geq 1 hasAttribute.Attribute \\
&Attribute \equiv \geq 1 valueAtt.Value \\
&CtrlNode \equiv \leq 1 valueAtt.Value \\
&ActNode \equiv == 1 valueAtt.Value \\
&Value \equiv valueRef.Value
\end{aligned}
$$

Fig. 1: CPN ontology expressed in a description logic

attributes of individuals); **Value** defined for all subsets of $I_1 \times I_2 \times \ldots \times I_n$ where I_i is a set of individuals.

Properties between the concepts in the CPN Ontology are also specified in Figure 1. For example, the concept $CPNOnt$ is defined with three properties, including $hasPlace$, $hasTrans$ and $hasArc$. It can be glossed as 'The class $CPNOnt$ is defined as the intersection of: (i) any class having at least one property $hasPlace$ whose value is restricted to the class $Place$ and; (ii) any class having at least one property $hasTransition$ whose value is restricted to the class $Transition$ and; (iii) any class having at least one property $hasArc$ whose value is either restricted to the class $InputArc$ or the class $OutputArc$'.

4 Semantic Constraints for Business Processes

As mentioned previously, our work aims at representing workflow processes modelled with CPNs in a knowledge base. Therefore, in this Section, we focus on ensuring their quality by guaranteeing their semantic correctness.

4.1 Definition of Semantic Constraints

By taking account domain experts in support of modellers at build time, a set of semantic constraints is specified, which then is used to develop a corresponding workflow. According to [13], there are two fundamental kinds of semantic constraints, including mutual exclusion constraints and dependency constraints. For interdependent tasks, e.g., the presence of task A indicates that task B must be included, however, task B can be executed while task A is absence. In fact, there may exist tasks that are coexistent. This refers to the coexistence constraints. Consequently, we propose three basic types: mutual exclusion constraints, dependency constraints and coexistence constraints.

Definition 1 (Semantic Constraint).

Let T be a set of tasks. A semantic constraint:

$c = (constraintType, appliedTask, relatedTask, order, description, [Equivalence])$ *where:*

- $constraintType \in \{mExclusion, dependency, coexistence\}$;
- $appliedTask \in T$;
- $relatedTask \in T$;
- $order \in \{before, after, concurrence, notSpecified\}$;
- *description is an annotation of the constraint;*
- *Equivalence is a set of tasks which are equivalent to task appliedTask.*

In Definition 1, the first parameter *constraintType* denotes the type of a semantic constraint. Each value of *constraintType* refers to the relationship between the executions of the source task denoted by the second parameter *appliedTask* and the target task denoted by the third parameter *relatedTask*. Parameter *order* specifies the order between the source and target tasks in a process model. The first four parameters are very important when defining a semantic constraint. The fifth parameter, *description*, is used for describing the constraint. *Equivalence* is an optional parameter, which contains a set of tasks (if any) being equivalent to the source task.

Let us continue the example of a process for the Order Management activity. The process is determined as follows: After receiving an order, two tasks have to do in parallel are *authenticate client* and *check availability*. If both of these tasks result "true", the order is accepted and an order confirmation is sent out. Otherwise, an order refusal is sent out. Some semantic constraints of the process are formed as follows:

```
c1 = (dependency, authenticate client, receive request, before,
receiving an order has to be performed before authenticating
client, {authenticate purchaser});
c2 = (dependency, check availability, receive request, before,
receiving an order has to be performed before checking
availability);
c3 = (coexistence, authenticate client, check availability,
concurrence, client authentication and checking availability
are performed in parallel);
c4 = (dependency, evaluate results, authenticate client, before,
evaluating the results obtained from the relevant departments);
c5 = (dependency, evaluate results, receive request, before,
receiving an order has to be performed before evaluating results
related to the order)
```

4.2 Checking implicit, redundant and conflicting semantic constraints

A workflow process is designed based upon the specified semantic constraints. However, when defining those constraints, there may occur implicit, redundant or conflicting semantic constraints.

Note that a combination of two or more constraints can constitute some new constraints. This is demonstrated by the *order* parameter in Definition 1. As mentioned above, this parameter indicates the execution order of a source task and a target task. Consider T_1, T_2, T_3, instances of tasks, we identify the following properties which are associative, symmetric, transitive and/or commutative presented in Table 1.

Table 1: Algebra properties of the order parameter

Name		Expression
Association	(1)	$(T_1\ concurrence\ T_2)\ concurrence\ T_3 = T_1\ concurrence\ (T_2\ concurrence\ T_3)$
	(2)	$(T_1\ notSpecified\ T_2)\ notSpecified\ T_3 = T_1\ notSpecified\ (T_2\ notSpecified\ T_3)$
Symmetrization	(1)	$T_1\ before\ T_2 = T_2\ after\ T_1$
Transitivity	(1)	$T_1\ before\ T_2, T_2\ before\ T_3 \rightarrow T_1\ before\ T_3$
	(2)	$T_1\ after\ T_2, T_2\ after\ T_3 \rightarrow T_1\ after\ T_3$
	(3)	$T_1\ concurrence\ T_2, T_2\ concurrence\ T_3 \rightarrow T_1\ concurrence\ T_3$
	(4)	$T_1\ concurrence\ T_2, T_1\ before\ T_3 \rightarrow T_2\ before\ T_3$
	(5)	$T_1\ concurrence\ T_2, T_1\ after\ T_3 \rightarrow T_2\ after\ T_3$
Commutativity	(1)	$T_1\ concurrence\ T_2 = T_2\ concurrence\ T_1$
	(2)	$T_1\ notSpecified\ T_2 = T_2\ notSpecified\ T_1$

These properties are used for inferring implicit semantic constraints. As a result, detecting implicit constraints plays a crucial role in the elimination of redundant constraints. In addition, conflicting constraints will lead to undesirable results. Therefore, it is necessary to resolve them before they can be applied.

Let us consider three constraints presented in Subsection 4.1, including $c1, c4$ and $c5$. According to transitivity property (1) in Table 1, a new constraint, named $c1\_4$, can be inferred from constraints $c1$ and $c4$, where:

```
c1_4=(dependency, evaluate results, receive request, before,
receiving an order has to be performed before authenticating
client and evaluating the results obtained from the relevant
departments)
```

Comparing constraint $c1\_4$ to constraint $c5$, their first four values are the same, hence constraint $c5$ is redundant. As a result, constraint $c5$ has to be removed.

Because of the different complexity of each semantic constraint set, we need an algorithm to resolve issues related to redundancy and conflict semantic constraints. The procedure for validating the constraint set will stop as soon as it

detects a couple of conflicting constraints and a message is generated to notify the user. In addition, if there exist any redundant constraints, they will be removed. In our algorithm in Figure 2, the boolean function $conflict$ is used for checking the conflict between two constraints, e.g., it returns $true$ if they are conflicting, otherwise, it returns $false$. The function $infer$ is used for inferring implicit constraints. The time complexity of the algorithm is $O(n^3)$ where n is the number of semantic constraints. To provide a representation of semantic

```
Input: Initial semantic set vector C
Output: Sound semantic constraint set vector C
SCValidation (C: Semantic_Constrait_Set)
1:  { n = C.size;
2:  For (i = 1; i<=n-1; i++)
3:   For (j = i+1; j<=n; j++)
4:     If (conflict(C[i],C[j]))
5:        { Print ‘‘Constraint C[i] conflicts with constraint C[j]’’;
6:          Break ;   }
7:     Else If (!empty(infer(C[i],C[j])))
8:         //If there exists an implicit constraint
9:        { cij = infer(C[i],C[j]);
10:         For (k = j+1; k<=n; k++)
11:           If (conflict(cij,C[k]))
12:         { Print ‘‘The implicit constraint inferred from C[i] and C[j]
13:           conflicts with constraint C[k]’’ ;
14:         Break ;  }
15:       Else If (compare(cij,C[k]))
16:        {  C.Remove(C[k]) ; //Remove redundant constraint C[k]}
18: }
```

Fig. 2: Algorithm for validating the semantic constraint set

constraints related to process elements, in next Subsection, we will describe an approach for constructing a semantic conformance-oriented ontology.

4.3 Development of a Semantic conformance-oriented Ontology

Our work aims at representing processes modelled with CPNs in a knowledge base. Therefore, to provide a representation of semantic constraints related to process elements, we develop an approach for constructing a new ontology. This ontology is oriented to semantic conformity checking in workflow processes. We focus on formalizing the concepts/relations corresponding to the knowledge that is required by model elements.

The following keystones to transform a set of semantic constraints into an OWL DL ontology:

- Each semantic constraint c is mapped to an instance of $owl : Class$.

- *appliedTask* and *relatedTask* are mapped to two instances of $owl : Class$. The $rdfs : subClassOf$ property is used to state that these classes is a subclass of the constraint class.
- Each value of *constraitType* or *order* is defined as an instance of the built-in OWL class $owl : ObjectProperty$.
- *description* is defined as an instance of the built-in OWL class $owl : Datatype\ Property$;
- Each value in the set *Equivalence* is mapped to an instance of $owl : Class$. The built-in property $owl : equivalentClass$ is used to link every class description of these classes to the class description of $appliedTask$.

In the upcoming Section, we will discuss about the integration of a semantic conformance-oriented ontology (domain knowledge) and the CPN Ontology to create workflow processes.

5 Creation of Correspondences Between Ontologies

We rely on ontology mapping techniques for matching semantics between ontologies, i.e., the CPN Ontology and Domain Ontology (a semantic conformance-oriented ontology). In our case, the articulation of two ontologies are used not only for creating semantically workflow processes, but also for verifying their correctness.

We now define our use of the term "mapping": Consider two ontologies, O_1 and O_2. Mapping of one ontology with another is defined as bringing ontologies into mutual agreement in order to make them consistent and coherent. It means that for a concept or a relation in ontology O_1, we try to find the same intended meaning in ontology O_2; For an instance in ontology O_1, we find the same instance in ontology O_2.

Definition 2 (Mapping related to the *before* property).
We define a mapping for all instances I_C of a semantic constraint in which the order between the instance of class appliedTask, named $task_a$, and the instance of class relatedTask, named $task_b$, is indicated by the object property before. The type of instance I_C is either dependency or coexistence. A set of correspondences is determined as follows:

- *Each instance of class appliedTask or relatedTask is mapped into an instance of class Transition (expressing activity node).*
- *There exists a firing sequence $t_1 t_2 \dots t_n$, where t_1, t_n are the instances of class Transition corresponding to instances $task_a$ and t_b respectively, $t_a = t_1$, $t_b = t_n$ $n \geq 2$.*

Definition 3 (Mapping related to the *concurrence* property).
We define a mapping for all instances I_C of a semantic constraint in which the order between the instance of class appliedTask, named $task_a$, and the instance of class relatedTask, named $task_b$, is indicated by the object property concurrence. The type of instance I_C is coexistence. A set of correspondences is determined as follows:

- *Each instance of class appliedTask or relatedTask is mapped into an instance of class Transition (expressing activity node).*
- *Two instances of class transitions which correspond to instance* $task_a$ *and instance* $task_b$ *can be enabled at the same time.*

It is important to underline that object property $before$ is the symmetrical property of object property $after$. Consequently, we do not define a mapping related to the $after$ property.

By continuing the process schema for the order management activity in Section 4, Figure 3 shows the mapping of some instances between two onotologies, CPN Ontology and Semantic Conformance-oriented Ontology.

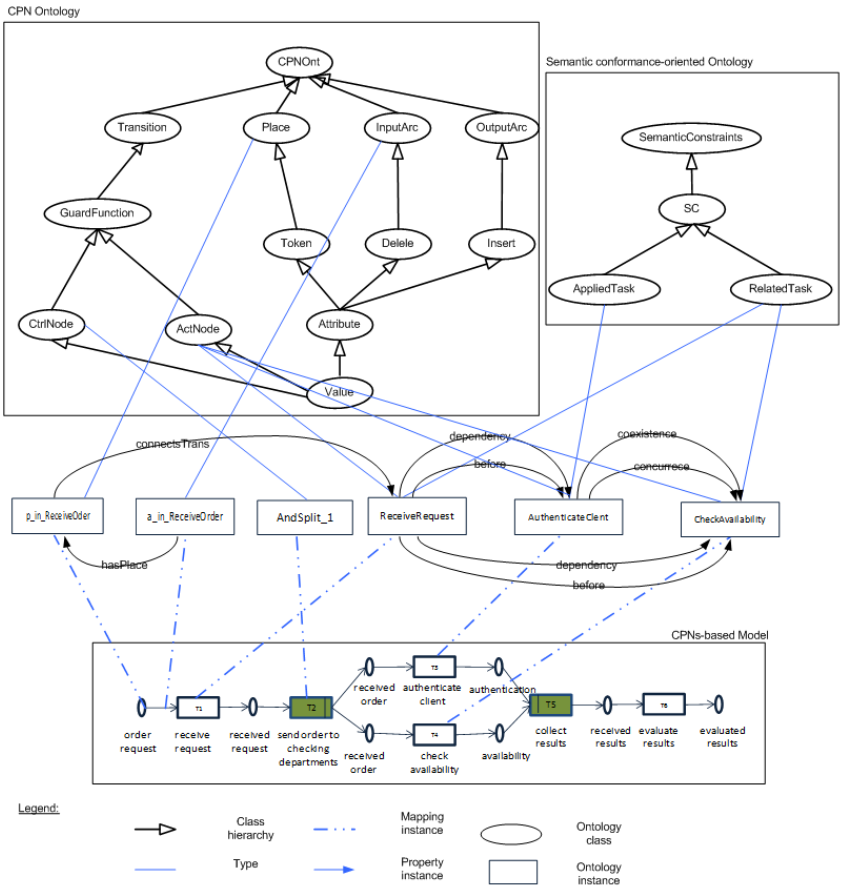

Fig. 3: An example of ontology mapping (excerpt)

We have introduced the formal definition of semantic constraints and illustrated how to model a workflow process with CPNs based on specified semantic

constraints. Note that concrete workflow processes are represented in RDF syntax. Moreover, to develop or modify a workflow process, manipulation operations [20] (e.g., inserting a new element) are required. Therefore, it is necessary to verify workflow processes at design time before they are put into use.

6 Semantic Verification Issues

By using the algorithm presented in Section 4, the sets of specified semantic constraints are checked redundant and conflicting. Hence, we here pay attention to the research question relating to semantic verification: Is the behavior of the individual activities satisfied and conformed with the control flow? To answer this question, we address the following semantic verification issues:

- Are there activities whose occurrences are mutual exclusion, but that may be executed in parallel or in sequence?
- Are there activities whose executions are interdependent, but that may be carried out in choice or in parallel?
- Are there activities whose occurrences are coexistent, but that may be executed in choice?
- Are there any couples of activities whose order executions are defined as one before the other, but that may be executed in the opposite?
- Are there any couples of activities whose order executions are defined as one after the other, but that may be executed in the opposite order?

Because concrete workflows are stored in RDF syntax, we rely on the CORESE [23] semantic search engine for answering SPARQL queries asked against an RDF knowledge base. We initiate SPARQL queries to verify whether workflow processes contain semantic errors or not. SELECT query form is chosen for this work. After a SELECT keyword, the variables are listed that contain the return values. And in the WHERE clause, one or more graph patterns can be specified to describe the desired result.

The following query[1] relating to the third verification issue is used to query if the model contains 'any pairs of activities whose occurrences are coexistence but that may be executed in choice'. The properties $h : coexistence$ and $h : concurrence$ defined in the first ontology indicate the semantic constraint between activities $?t1$ and $?t2$. On the other hand, the other properties defined in the second ontology which represent these activities restricted to the control flow perspective. By applying this query to the workflow example depicted in Figure 3, the result is empty.

The sample query does not only demonstrate that the SPARQL query language is able to check the semantic correctness of workflow processes, but also the usage of terminological background knowledge provided by the semantic conformance-oriented ontology and CPN Ontology.

[1] The prefix is assumed as:
$PREFIX h :< http://www.semanticweb.org/CPNWF\# >$

Moreover, by representing CPNs-based business processes with OWL DL ontology we can also verify the soundness of models. This means that we can check syntactic errors (for example, deadlocks, infinite cycles and missing synchronization, etc.) by the SPARQL query language.

```
SELECT ?t1 ?t2 WHERE
{
?t1 rdf:type h:Transition
?t2 rdf:type h:Transition
?t3 rdf:type h:Xor-split
?t4 rdf:type h:Xor-join
?t1 h:coexistence ?t2
?t2 h:concurrence ?t1
?t3 h:connectsPlace/h:connectsTrans ?t1
?t3 h:connectsPlace/h:connectsTrans ?t2
?t1 h:connectsPlace/h:connectsTrans ?t4
?t2 h:connectsPlace/h:connectsTrans ?t4
FILTER (?t1!=?t2)
}
```

7 Conclusion

This paper presents a formal method for describing semantic constraints that are used to generate workflow processes. First, we propose a formal definition of semantic constraints. Then we describe an algorithm for detecting redundant and conflicting semantic constraints. To integrate the domain knowledge used for annotating the process elements, we develop a semantic conformance-oriented ontology. This ontology is then matched with the CPN Ontology (a representation of CPNs with OWL DL). The mapping is to enable workflow processes which can be verified at the semantic level and also syntactic level. Furthermore, we demonstrate that the SPARQL query language is able to check the correctness of concrete workflow processes represented in RDF syntax. This ensures error-free workflow processes at build-time.

We know that verifying workflow processes at build-time is not enough to guarantee workflows can be executed correctly. The correctness of workflow execution must also be checked. Therefore, in future work, we plan to develop a run-time environment for validating concrete workflows.

References

1. Ellis, C.A., Nutt, G.J.: Modeling and enactment of workflow systems. In: Application and Theory of Petri Nets. (1993) 1–16
2. van der Aalst, W.M.P.: The application of petri nets to workflow management. Journal of Circuits, Systems, and Computers **8**(1) (1998) 21–66

3. Barros, A.P., ter Hofstede, A.H.M., Proper, H.A.: Essential principles for workflow modelling effectiveness. In: PACIS. (1997) 15
4. Koschmider, A., Oberweis, A.: Ontology based business process description. In: EMOI-INTEROP, Springer (2005) 321–333
5. Fellmann, M., Thomas, O., Busch, B.: A query-driven approach for checking the semantic correctness of ontology-based process representations. In: BIS. (2011) 62–73
6. W3C: Sparql 1.1 query language. http://www.w3.org/TR/sparql11-query/ (March 2013) W3C Recommendation.
7. van der Aalst, W.M.P.: Verification of workflow nets. In: ICATPN. (1997) 407–426
8. Verbeek, H., Basten, T., van der Aalst, W.: Diagnosing workflow processes using woflan. The computer journal **44** (1999) 246–279
9. Bi, H.H., Zhao, J.L.: Applying propositional logic to workflow verification. Information Technology and Management **5**(3-4) (2004) 293–318
10. Wainer, J.: Logic representation of processes in work activity coordination. In: Proceedings of the 2000 ACM Symposium on Applied Computing - Volume 1. SAC '00, New York, NY, USA, ACM (2000) 203–209
11. Sadiq, W., Maria, Orlowska, E.: Analyzing process models using graph reduction techniques. Information Systems **25** (2000) 117–134
12. Lu, S., Bernstein, A.J., Lewis, P.M.: Automatic workflow verification and generation. Theor. Comput. Sci. **353**(1-3) (2006) 71–92
13. Ly, L.T., Rinderle, S., Dadam, P.: Integration and verification of semantic constraints in adaptive process management systems. Data Knowl. Eng. **64**(1) (2008) 3–23
14. Kumar, A., Yao, W., Chu, C.H., Li, Z.: Ensuring compliance with semantic constraints in process adaptation with rule-based event processing. In: RuleML. (2010) 50–65
15. Ly, L.T., Rinderle-Ma, S., Göser, K., Dadam, P.: On enabling integrated process compliance with semantic constraints in process management systems - requirements, challenges, solutions. Information Systems Frontiers **14**(2) (2012) 195–219
16. Thomas, O., Fellmann, M.: Semantic process modeling - design and implementation of an ontology-based representation of business processes. Business & Information Systems Engineering **1**(6) (2009) 438–451
17. Weber, I., Hoffmann, J., Mendling, J.: Beyond soundness: on the verification of semantic business process models. Distributed and Parallel Databases **27**(3) (2010) 271–343
18. Gasevic, D., Devedzic, V.: Interoperable petri net models via ontology. Int. J. Web Eng. Technol. **3**(4) (2007) 374–396
19. Sebastian, A., Tudorache, T., Noy, N.F., Musen, M.A.: Customizable workflow support for collaborative ontology development. In: 4th International Workshop on Semantic Web Enabled Software Engineering (SWESE) at ISWC 2008. (2008)
20. Nguyen, T.H.H., Le-Thanh, N.: An ontology-enabled approach for modelling business processes. In: Beyond Databases, Architectures, and Structures. Volume 424 of Communications in Computer and Information Science. Springer International Publishing (2014) 139–147
21. Kristensen, L.M., Christensen, S., Jensen, K.: The practitioner's guide to coloured petri nets. STTT **2**(2) (1998) 98–132
22. W3C: Owl web ontology language reference. http://www.w3.org/TR/owl-ref/ (2004) W3C Recommendation.
23. Corby, O., et al.: Corese/kgram. https://wimmics.inria.fr/corese