Ontology-Based Discovery of Workflow Activity Patterns

Diogo R. Ferreira[1], Susana Alves[1], Lucinéia H. Thom[2]

[1] IST – Technical University of Lisbon, Portugal
{diogo.ferreira,susana.alves}@ist.utl.pt
[2] Université Joseph Fourier, France / Institute of Informatics, UFRGS, Brazil
lucineia@inf.ufrgs.br

Abstract. Workflow activity patterns represent a set of recurrent behaviors that can be found in a wide range of business processes. In this paper we address the problem of determining the presence of these patterns in process models. This is usually done manually by the analyst, and it requires interpreting the process in terms of the semantics of those patterns. We describe an ontology-based approach to perform this discovery in an automated way. The approach makes use of an ontology, and a mapping between the elements in the given process and the classes in the ontology. A reasoner is then used to discover the patterns, and a SPARQL query is used to retrieve them. The approach is illustrated for a business process in a travel booking scenario.

Key words: Business Process Modeling, Workflow Activity Patterns, Ontology Engineering, Semantic Reasoning

1 Introduction

Business processes can be seen as being composed of a number of different patterns, which have already been thoroughly studied in the literature [1, 2]. There have been also attempts at explaining business processes by means of a single pattern, such action-workflow [3] or a basic transaction pattern [4]. In general, these patterns fulfill a double role of facilitating the understanding of processes on one hand, and on the other hand providing the building blocks from which new processes can be designed. Most of the previous work has therefore focused on identifying these building blocks and deciding which of them are most appropriate to capture the common structures of business processes.

Here we take a different viewpoint of assuming that these patterns have been already defined, and instead we focus on the problem of determining whether a given set of patterns is present in a given business process. In particular, we are interested in recognizing the presence of patterns by making use of the semantics of the business process, i.e. we are looking not only at the structural behavior of business processes, but especially at the *meaning* of the activities contained in a process. For example, if we know that a certain activity can be interpreted as an approval step, then it is possible that the process contains an approval pattern, which occurs very often in business processes.

We are dealing with so-called *workflow activity patterns* [5] which represent business functions that typically occur in every business process, such as *activity execution*, *decision making*, *notification*, *approval*, etc. These business functions cannot be identified solely by looking at the structure of a process; it is necessary to understand the purpose of each activity in order to decide whether it corresponds to a known business function. In addition, we cannot say that the process contains an approval pattern just because it has an approval step; all the required elements of the approval pattern should be present in order to consider that the process contains such pattern. Section 2 provides a summary of these patterns.

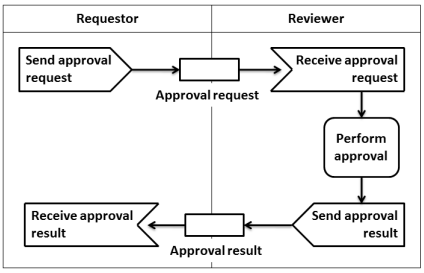
Discovering workflow activity patterns in business processes is typically done manually by the process analyst, and it is not a trivial task since the purpose and use of any given activity can be given different interpretations. Also, if such pattern analysis must be conducted over a large repository of process models, it can become a tedious and error-prone task. Our goal is to provide an automated means which can significantly accelerate the discovery of patterns in process models and relieve the analyst from having to do an exhaustive manual search. Since, to a large extent, such discovery is based on semantics, we turn to an ontology- and reasoning-based approach, as described in Section 3.

Throughout the paper we use the example of a travel booking process introduced in [5]. The experimental evaluation of the proposed approach in more realistic and complicated process models faces a number of additional challenges that we are unable to address here. However, by describing the principles and implementation of the approach, the reader will hopefully get a sense for the potential of using ontologies and automated reasoning to address challenging problems in the area of Business Process Management, especially those which, like the problem addressed here, must rely on semantics to a large extent.

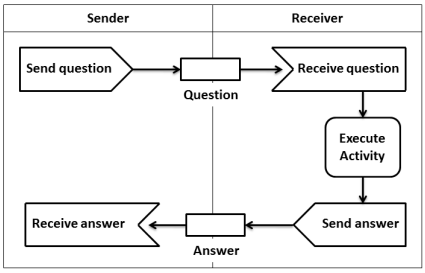
2 Workflow Activity Patterns

Workflow activity patterns (WAPs) [5] are common structures that can be found in a variety of business processes. These structures involve control-flow constructs as well as interactions between participants and also the semantics of the activities being performed. Our starting point will be the seven WAPs as defined in [5]. These comprise the following behaviors:

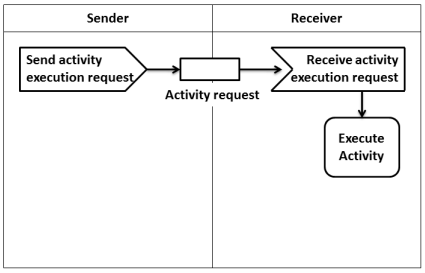
1. *Approval*: An object (e.g. a document) has to be approved by some organizational role. A requestor sends the approval request to a reviewer, who performs the approval and returns a result.
2. *Question-Answer*: When performing a process, an actor might have a question before working on the process or on a particular activity. This pattern allows to formulate such question, to identify an organizational role who is able to answer it, to send the question to the respective actor filling this role, and to wait for response.



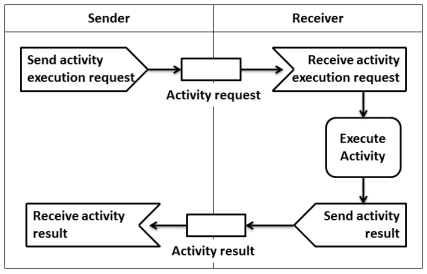
WAP1: Approval



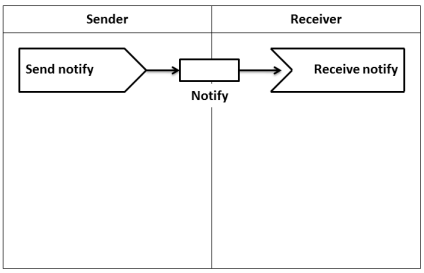
WAP2: Question-Answer



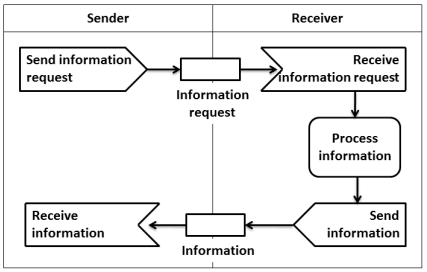
WAP3: Unidirectional Performative



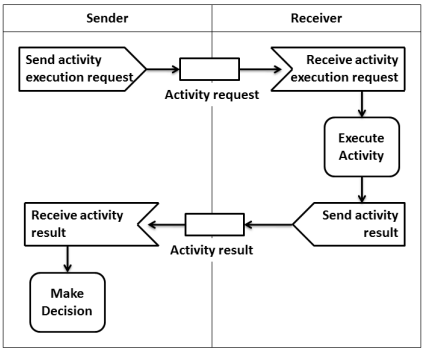
WAP4: Bidirectional Performative



WAP5: Notification



WAP6: Information Request



WAP7: Decision

Fig. 1. Simplified versions of the seven WAPs defined in [5]

3. *Unidirectional Performative*: A sender requests the execution of a particular activity from a receiver (e.g., a human or a software agent) involved in the process. The sender continues execution of his part of the process immediately after having sent the request.
4. *Bidirectional Performative*: A sender requests the execution of a particular activity from another role (e.g., a human or a software agent) involved in the process. The sender waits until the receiver notifies him that the requested activity has been performed.
5. *Notification*: The status or result of an activity execution is communicated to one or more process participants.
6. *Information Request*: An actor requests certain information from a process participant. He continues process execution after having received the desired information.
7. *Decision*: During process enactment, the performance an activity is requested. Depending on the result of the requested activity, the process continues execution with one or several branches. This pattern allows to include a decision activity with connectors to different subsequent execution branches (each of them associated with a specific transition condition). Exactly those branches are selected for execution whose transition condition evaluates to true.

Figure 1 provides a summary of these workflow activity patterns in graphical form. The patterns are composed of certain elements, namely *signals* (send and receive), *activities* (e.g. "Perform approval") and *messages* (e.g. "Approval request"). For example, WAP1 begins by a send signal with an approval request message; then there is a receive signal for that same message; then an activity to perform the approval; and finally the exchange of the approval result by another pair of send and receive signals.

For simplicity, we have deliberately omitted some elements from these patterns. For example, WAP2 as originally defined in [5] contains additional activities before "Send question", namely an activity "Describe question" and another activity "Identify role habilities". These are elements that could be used, in effect, to distinguish WAP2 from other patterns. By omitting some elements, the patterns become very similar in terms of structure, as can be seen in Figure 1. However, there are some clear differences in purpose and semantics between them, and it is precisely these semantics, rather than structure, that we will use to discover them in business process models.

3 Ontology-Based Approach

Figure 2 shows an example of a travel booking process that has been modeled using the same kind of elements that were used to define the seven workflow activity patterns. However, the process makes use of its own vocabulary that is specific to this application domain. Our goal is to understand the semantics of each activity and to reason about these elements in order to determine which

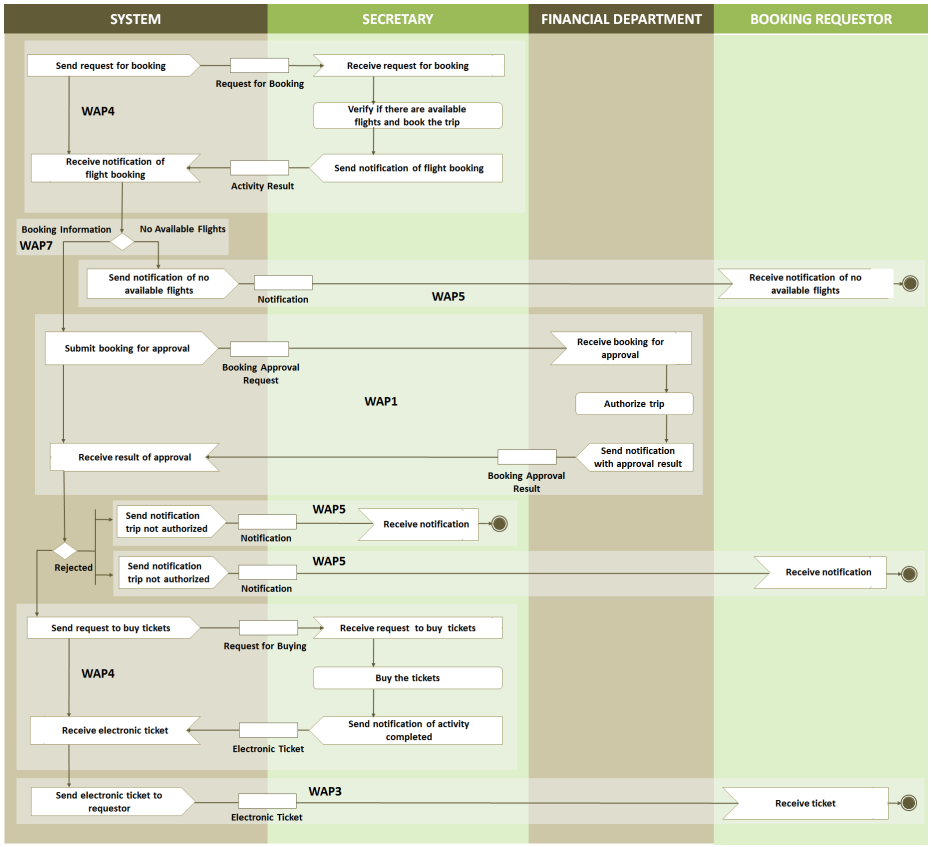


Fig. 2. Travel booking example (adapted from [5])

patterns are present in this process. Note that Figure 2 already includes an indication of the patterns that were found manually by an analyst. Our goal is to discover these patterns automatically and compare the results.

3.1 Defining the WAP Ontology

In order to reason about concrete examples such as the one depicted in Figure 2, we need an *ontology* that provides a description of the patterns to be discovered, and we need a *mapping* of the elements in the given process to the concepts defined in that ontology. For example, one should understand that the shape "Send request for booking" in Figure 2 is in effect a send signal with an activity request message as in WAP4; one should also realize that "Authorize trip" corresponds to a "Perform approval" activity as in WAP1; and so on. In order to do this, one needs to have an ontology that specifies these pattern elements.

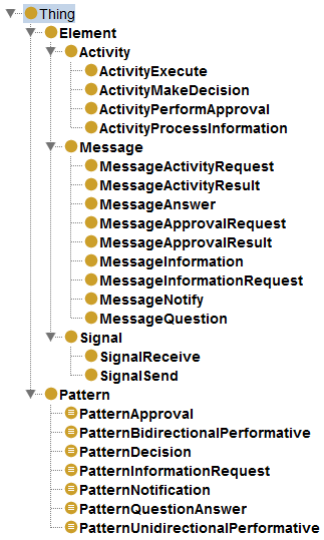


Fig. 3. Class hierarchy for the WAP ontology.

Figure 3 shows the class hierarchy for the WAP ontology that has been developed in this work, as it appears in Protégé[1]. Basically, there are two top-level classes, Element and Pattern, with Element being the superclass for the various pattern elements, and Pattern being the superclass for the definitions of the several workflow activity patterns. The rationale for this ontology can be summarized as follows:

- Each Pattern is defined as containing certain elements of the classes Signal and Activity. For this purpose we define the object property hasElement with domain Pattern and range Element. Example: PatternApproval hasElement ActivityPerformApproval.
- Each Signal has a certain kind of Message and for this purpose we define the object property hasMessage with domain Signal and range Message. Example: PatternApproval hasElement (SignalSend and (hasMessage MessageApprovalRequest)).

Each subclass of Pattern is defined by an equivalent class expression that specifies all the elements that the pattern contains. The complete definition for WAP1 is as follows:

```
PatternApproval ≡ Pattern
                  and (hasElement some (SignalSend
                                        and (hasMessage
                                             some MessageApprovalRequest)))
                  and (hasElement some (SignalReceive
                                        and (hasMessage
                                             some MessageApprovalRequest)))
```

[1] Protégé is available at: `http://protege.stanford.edu`

```
and (hasElement some ActivityPerformApproval)
and (hasElement some (SignalSend
                      and (hasMessage
                           some MessageApprovalResult)))
and (hasElement some (SignalReceive
                      and (hasMessage
                           some MessageApprovalResult)))
```

In general, a process may contain many elements, with only some of them matching a given pattern. Therefore, we make use of the keyword some, meaning that it is necessary for a pattern/signal to have at least one element/message of that kind, but possibly more. The definitions for the remaining patterns are analogous, and they are omitted for brevity; those definitions are similar to the one above, but make use of different elements. In particular, the definitions for WAP3 and WAP5 are shorter, while WAP7 has an additional activity.

On a final note about the ontology, we should mention that this is not the first time that an ontology for workflow activity patterns has been defined. In [6] the authors make use of a WAP ontology for the purpose of supporting process modeling; in this case the ontology describes the patterns and the relationships between them in order to produce recommendations about the possible use of other patterns in the same model; ultimately, it is the user who decides whether a given pattern should be inserted in the model. Here, we have built a different WAP ontology for the specific purpose of being able to infer which patterns are present in a given process model; we have therefore focused more on specifying the building blocks (elements) of these patterns, and on how these patterns are defined in terms of the elements they contain.

3.2 Mapping of Model Elements to Ontology Classes

While the ontology above defines the classes, the process model contains the elements that will be mapped as *individuals* of those classes. For example, the first shape "Send request for booking" in Figure 2 corresponds to two elements: a signal and a message. The signal is an individual of SignalSend and the message is an individual of MessageActivityRequest. We have therefore:

```
Element1 : SignalSend
Element2 : MessageActivityRequest
Element1 hasMessage Element2
```

As another example, the shape "Authorize trip" is an individual of ActivityPerformApproval, so we could have:

```
Element3 : ActivityPerformApproval
```

Now, the whole process is represented as an individual of Pattern so that from the above we would have:

Process1 : Pattern
Process1 hasElement Element1
Process1 hasElement Element3

Note that there is no need to assert Process1 hasElement Element2 since Element2 is a message and it is associated with Element1 via the hasElement property.

Once the mapping between the shapes in the model and the classes in the ontology is known, creating these individuals is straightforward and can be done automatically. Then a reasoner can be invoked to infer the patterns that the process contains.

However, the critical point is precisely in creating the mapping, e.g. knowing that "Send request for booking" corresponds to two classes (SignalSend and MessageActivityRequest) and "Authorize trip" corresponds to ActivityPerformApproval. This mapping must be done manually by the analyst, and it is equivalent to annotating the model shapes with classes from the ontology. This can be achieved in a similar way to other approaches that involve semantic annotation of business processes [7, 8, 9]. Still, creating such mapping is made difficult by the fact that the shapes in a process model use a domain-specific vocabulary and are often labeled in different ways. To facilitate this task, it would be desirable to have the shapes in a process model labeled in a consistent way, such as using *verb-object* style as proposed in [10].

For the process in Figure 2 we have the following mapping:

Send request for booking :: SignalSend MessageActivityRequest
Receive request for booking :: SignalReceive MessageActivityRequest
Verify if there are available flights and book the trip :: ActivityExecute
Send notification of flight booking :: SignalSend MessageActivityResult
Receive notification of flight booking :: SignalReceive MessageActivityResult
Send notification of no available flights :: SignalSend MessageNotify
Receive notification of no available flights :: SignalReceive MessageNotify
Submit booking for approval :: SignalSend MessageApprovalRequest
Receive booking for approval :: SignalReceive MessageApprovalRequest
Authorize trip :: ActivityPerformApproval
Send notification with approval result :: SignalSend MessageApprovalResult
Receive result of approval :: SignalReceive MessageApprovalResult
Send notification trip not authorized :: SignalSend MessageNotify
Receive notification :: SignalReceive MessageNotify
Send request to buy tickets :: SignalSend MessageActivityRequest
Receive request to buy tickets :: SignalReceive MessageActivityRequest
Buy the tickets :: ActivityExecute
Send notification of activity completed :: SignalSend MessageActivityResult
Receive electronic ticket :: SignalReceive MessageActivityResult
Send electronic ticket to requestor :: SignalSend MessageNotify
Receive ticket :: SignalReceive MessageNotify

Provided with this mapping, the individuals and their properties are generated automatically. For each class in the mapping, a new individual is created

from that class. If the first class is a Signal and the second class is a Message, we add the property hasMessage which relates those two individuals. Finally, we create an individual of Pattern to represent the whole process, and we associate all signals and activities to the process via the property hasElement.

3.3 Pattern Discovery through Reasoning

Through the use of reasoning, it is possible to obtain additional statements that can be inferred from the available classes and individuals. The type of inference we will be most interested in is class membership. As explained above, each WAP is defined by an equivalent class expression that specifies the elements that the pattern contains. If a process has all the elements that satisfy a given pattern expression, then the process will become a member of that class (a subclass of Pattern). In general, a process may end up as a member of several classes, meaning that one can find in the process all the elements required by those patterns.

As an example, let us consider the following excerpt of the travel booking process:

Submit booking for approval :: SignalSend MessageApprovalRequest
Receive booking for approval :: SignalReceive MessageApprovalRequest
Authorize trip :: ActivityPerformApproval
Send notification with approval result :: SignalSend MessageApprovalResult
Receive result of approval :: SignalReceive MessageApprovalResult

These will result in the following individuals being created:

Element1 : SignalSend
Element2 : MessageApprovalRequest
Element1 hasMessage Element2
Element3 : SignalReceive
Element4 : MessageApprovalRequest
Element3 hasMessage Element4
Element5 : ActivityPerformApproval
Element6 : SignalSend
Element7 : MessageApprovalResult
Element6 hasMessage Element7
Element8 : SignalReceive
Element9 : MessageApprovalResult
Element8 hasMessage Element9
Process1 : Pattern
Process1 hasElement Element1
Process1 hasElement Element3
Process1 hasElement Element5
Process1 hasElement Element6
Process1 hasElement Element8

A semantic reasoner is then able to infer the following statements:

```
Process1 rdf:type Thing
Process1 rdf:type PatternApproval
```

The process is a member of Thing since it is a Pattern and a Pattern is a subclass of Thing. The reasoner is also able to infer that the process is a member of PatternApproval since, by the elements it contains, it satisfies the expression for that class.

It should be noted that even before the individuals are created, invoking a reasoner on the WAP ontology produces the following statements:

```
PatternBidirectionalPerformative rdfs:subClassOf PatternUnidirectionalPerformative
PatternDecision rdfs:subClassOf PatternBidirectionalPerformative
```

This can be easily understood by inspection of Figure 1. In fact, WAP4 contains all the elements of WAP3 and therefore WAP4 satisfies the definition of WAP3. The same happens with WAP7 and WAP4; WAP7 extends WAP4 and therefore it fits the definition of WAP4. This means that any process that contains WAP4 will also be listed as containing WAP3, and any process containing WAP7 will contain WAP4, and therefore WAP3 as well.

3.4 Retrieving the Patterns with SPARQL

From the WAP ontology and the individuals created from a given process, the reasoner is able to produce a large number of statements. Not all of these statements will be equally interesting. For example, knowing that a process is a Thing is trivial; also, if a process contains both WAP3 and WAP4, the most interesting statement is that it contains WAP4, since we know that any process that contains WAP4 also contains WAP3. In general, we are interested in class memberships that are closer to the leafs of the class hierarchy, as this represents more specific knowledge about the process and the patterns it contains.

In order to retrieve the patterns that a process contains, we use the following SPARQL query:

```
1: PREFIX wap: ...
2: PREFIX rdf: ...
3: PREFIX rdfs: ...
4: SELECT ?pattern WHERE { wap:Process1 rdf:type ?pattern .
5:                         ?pattern rdfs:subClassOf wap:Pattern .
6:                         FILTER (?pattern != wap:Pattern) .
7:                         OPTIONAL { ?pattern2 rdfs:subClassOf ?pattern .
8:                                    wap:Process1 rdf:type ?pattern2 }
9:                         FILTER (!bound(?pattern2)) }
```

The query determines all class memberships of Process1 (line 4) where the class must be a subclass of Pattern (line 5). According to the OWL standard, a class is by definition a subclass of itself, so Pattern will also appear in the results;

we exclude this case with the filter expression in line 6. In lines 7-9 we exclude the case when the result indicates that the process contains both a pattern and a subclass of that pattern (as in WAP3 and WAP4). Lines 7-8 check if there is a subclass (e.g. WAP4) of the pattern (e.g. WAP3) that the process also contains. If so, then we are interested in the subclass (WAP4) rather than in the original class (WAP3). Line 9 excludes the result when there is such case.

Running this query on the travel booking example produces the following results: PatternApproval, PatternBidirectionalPerformative, and PatternNotification. Note that PatternUnidirectionalPerformative is excluded by lines 7-9 since PatternBidirectionalPerformative is a subclass of PatternUnidirectionalPerformative.

These results indicate that the process contains enough elements to satisfy the definition of three different patterns: WAP1, WAP4 and WAP5. However, when comparing these results with Figure 2, we note the absence of WAP7 and WAP3. This can be explained as follows:

- With regard to WAP7, this pattern is not detected since the process does not include an ActivityMakeDecision. The analyst considered that such activity is implicit in the diamond shape, but the element is absent from the mapping.
- With regard to WAP3, that part of the process is inferred as an instance of WAP5 rather than WAP3. This is because the message has been classified as MessageNotify in the mapping. However, it appears that the analyst originally thought that it was a MessageActivityRequest.

3.5 Implementation

The WAP ontology was developed and tested in Protégé 4.1 together with the Pellet Reasoner Plug-in[2]. We load the ontology and create the individuals in Java with the Jena framework[3] version 2.6.3. The Pellet reasoner[4] version 2.2.2 is invoked through Jena to perform reasoning over the ontology together with the individuals. The SPARQL query is also executed through Jena.

Basically, using Jena we load the ontology file created with Protégé into an ontology model (OntModel). Then we read a text file containing the mapping. For each class in the mapping we retrieve a class reference (OntClass) from the model, and then create an individual from that class using OntClass.createIndividual(). If the element is a signal then we also create and associate a message individual via the hasMessage property. Using the Pellet reasoner, we create an inference model (InfModel) and then run the SPARQL query over this new model. Iterating through the results provides the subclasses of Pattern contained in the process.

4 Conclusion

In this paper we have described an approach to automate the discovery of workflow activity patterns in process models by means of reasoning over an ontology.

[2] http://clarkparsia.com/pellet/protege/
[3] http://jena.sourceforge.net/
[4] http://clarkparsia.com/pellet/

In this ontology, the classes define the elements that each pattern contains, and the individuals represent the elements of a given process. Once the mapping between the process elements and the ontology elements is established, it is possible to invoke a semantic reasoner to determine which patterns are present in the process. This is done mainly by checking whether the process contains the necessary elements to fulfill the definition of each pattern.

In future work, we intend to develop the approach further in order to check that the elements are not only present, but that they also comply with the sequential behavior of workflow activity patterns. Meanwhile, we believe that the current approach can be useful to show the potential of using ontologies and automated reasoning to address challenging problems in the area of Business Process Management, especially those which, like the problem addressed here, rely on semantics to a large extent.

References

1. van der Aalst, W., ter Hofstede, A., Kiepuszewski, B., Barros, A.: Workflow patterns. Distributed and Parallel Databases **14**(1) (July 2003) 5–51
2. ter Hofstede, A., Dietz, J.: Generic recurrent patterns in business processes. In Weske, M., ed.: Business Process Management. Volume 2678 of Lecture Notes in Computer Science. Springer Berlin / Heidelberg (2003) 1018–1018
3. Medina-Mora, R., Winograd, T., Flores, R., Flores, F.: The action workflow approach to workflow management technology. In: Proceedings of the 1992 ACM conference on Computer-supported cooperative work. CSCW '92, ACM (1992) 281–288
4. Dietz, J.L.: The deep structure of business processes. Communications of the ACM **49** (May 2006) 58–64
5. Thom, L.H., Reichert, M., Iochpe, C.: Activity patterns in process-aware information systems: basic concepts and empirical evidence. International Journal of Business Process Integration and Management **4**(2) (2009) 93–110
6. Thom, L.H., Reichert, M., Chiao, C., Iochpe, C., Hess, G.: Inventing less, reusing more, and adding intelligence to business process modeling. In: Database and Expert Systems Applications. Volume 5181 of Lecture Notes in Computer Science. Springer (2008) 837–850
7. Born, M., Dörr, F., Weber, I.: User-friendly semantic annotation in business process modeling. In: Web Information Systems Engineering WISE 2007 Workshops. Volume 4832 of Lecture Notes in Computer Science. Springer (2007) 260–271
8. Zouggar, N., Vallespir, B., Chen, D.: Semantic enrichment of enterprise models by ontologies-based semantic annotations. In: Proceedings of the 12th International EDOC Conference Workshops, IEEE Computer Society (2008) 216–223
9. Filipowska, A., Kaczmarek, M., Stein, S.: Semantically annotated EPC within semantic business process management. In: Business Process Management Workshops. Volume 17 of LNBIP. Springer (2009) 486–497
10. Mendling, J., Reijers, H., Recker, J.: Activity labeling in process modeling: Empirical insights and recommendations. Information Systems **35**(4) (2010) 467–482