A Semantic Approach to the Discovery of Workflow Activity Patterns in Event Logs

Diogo R. Ferreira

IST – Technical University of Lisbon,
Avenida Prof. Dr. Cavaco Silva,
2744-016 Porto Salvo, Portugal
E-mail: diogo.ferreira@ist.utl.pt

Lucinéia H. Thom

Institute of Informatics,
Universidade Federal do Rio Grande do Sul,
91501-970 Porto Alegre, RS, Brazil
E-mail: lucineia@inf.ufrgs.br

Abstract: Workflow activity patterns represent a set of recurrent behaviours 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, who inspects the model and interprets its elements in terms of the semantics of those patterns. Here, we present an approach to perform this discovery based on the event log created during process execution. The approach makes use of an ontology and the semantic annotation of the event log in order to discover the patterns automatically by means of semantic reasoning. We illustrate the application of the proposed approach in a case study involving a purchase process implemented in a commercial workflow system.

Keywords: Business Process Modelling, Workflow Activity Patterns, Process Mining, Ontology Engineering, Semantic Reasoning

Reference to this paper should be made as follows: Ferreira, D.R. and Thom, L.H. (2012) 'A Semantic Approach to the Discovery of Workflow Activity Patterns in Event Logs', *Int. J. Business Process Integration and Management*, Vol. X, Nos. Y, pp.00–00.

Biographical notes: Diogo R. Ferreira is professor of information systems at the Technical University of Lisbon, and he is an active researcher in the field of business process management, particularly in the area of process mining.

Lucinéia H. Thom is a post-doc researcher at the Institute of Informatics of the Federal University of Rio Grande do Sul, Brazil. Her research interests are in the area of business process management and information technology for healthcare and homecare.

1 Introduction

Business processes can be seen as being composed of patterns, which have been thoroughly studied in the literature (van der Aalst et al., 2003a; ter Hofstede and Dietz, 2003). There have been also attempts at explaining business processes by means of a single pattern, such action-workflow (Medina-Mora et al., 1992) or a transaction pattern (Dietz, 2006). In any case, these patterns fulfil 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 common structures in 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 recognising the presence of patterns by making use of the semantics of the business process, i.e. we are looking not only at the structural behaviour 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.

We are dealing with the so-called *Workflow Activity Patterns* (WAPs) (Thom et al., 2009) 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 of the required elements of the approval pattern must 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 analyst, working over diagrams of the process model. Such analysis is non-trivial since the meaning and purpose of any given activity can be given different interpretations. Also, in addition to semantics, the analyst must be able to correctly interpret the control-flow in the model to ensure that the observed behaviour corresponds to the structure of the candidate patterns. Overall, this becomes a difficult and error-prone task. Our goal is to provide automated means to assist the analyst in the discovery of WAPs. Since, to a large extent, such discovery is based on semantics, we turn to an ontology-based approach, as described in Section 3. In addition, to facilitate the verification of the sequential behaviour in the control-flow, we make use of the event log generated during process execution, rather than original the process model, as explained in Section 4.

Throughout the presentation we use the example of a travel booking process introduced in (Thom et al., 2009). An experimental evaluation of the proposed approach in a more realistic process is provided in Section 5. By describing the principles, implementation and applications of the proposed approach, the reader will 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, rely on semantics to a large extent.

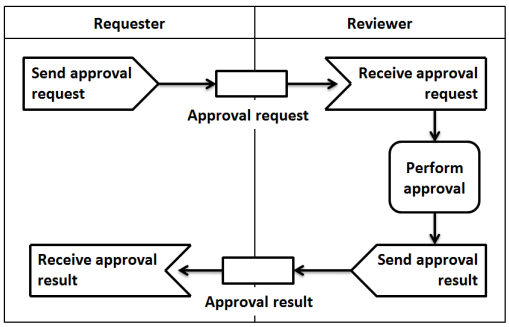
2 Workflow Activity Patterns

Workflow activity patterns (WAPs) (Thom et al., 2009) 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 (Thom et al., 2009). These comprise the following behaviors:

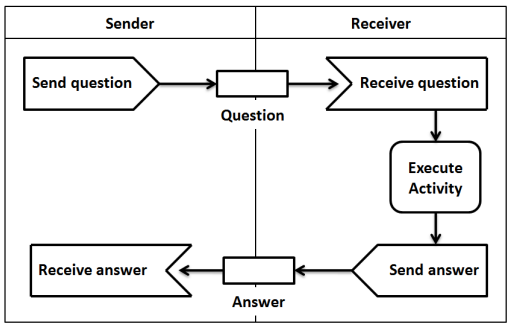
1. *Approval*: An object (e.g. a document) has to be approved by some organisational role. A requester sends the approval request to a reviewer, who performs the approval and returns a result.
2. *Question-Answer*: When performing a process, an actor might have a question before working on the process or on a particular activity. This pattern allows to formulate such question, to identify an organisational role who is able to answer it, to send the question to the respective actor filling this role, and to wait for response.
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 for a notification from the receiver 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 of an activity is requested. Depending on the result of the requested activity, the process continues execution with one or several branches. This pattern allows the inclusion of a decision activity which connects to different subsequent execution branches (each of them associated with a specific transition condition). Exactly those branches whose transition condition evaluates to true are selected for execution.

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; after, an activity to perform the approval; and finally the exchange of the approval result by another pair of send and receive signals.

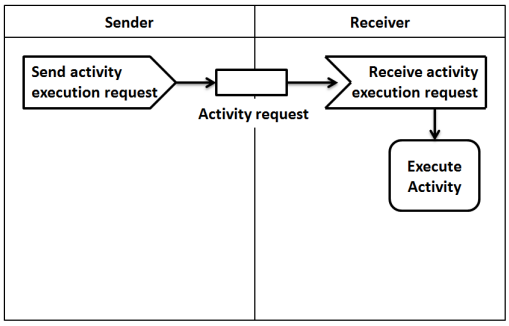
For simplicity, we have deliberately omitted some elements from these patterns. For example, WAP2 as originally defined in (Thom et al., 2009) contains additional activities before "Send question", namely an activity "Describe question" and another activity "Identify role habilities". These elements 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, in addition to structure, that we will use to discover them in business processes.



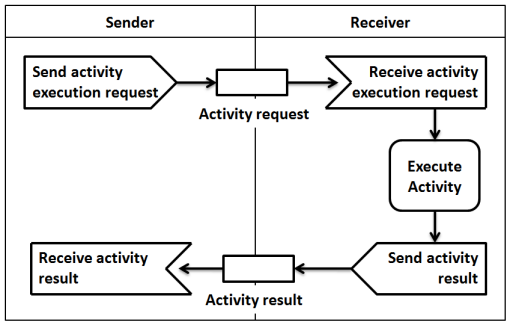
WAP1: Approval



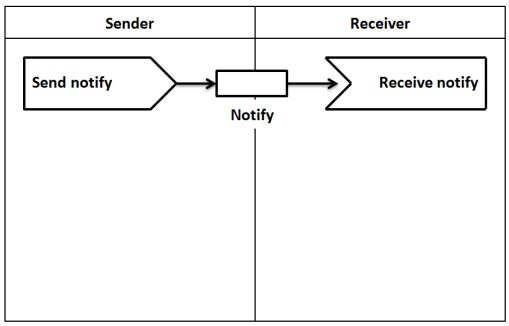
WAP2: Question-Answer



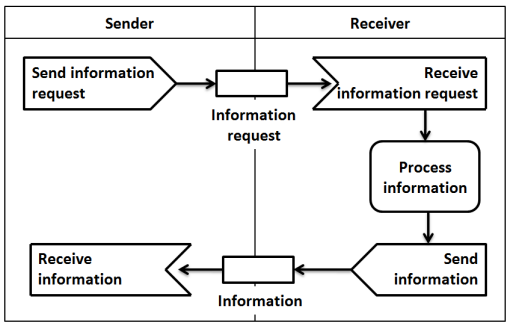
WAP3: Unidirectional Performative



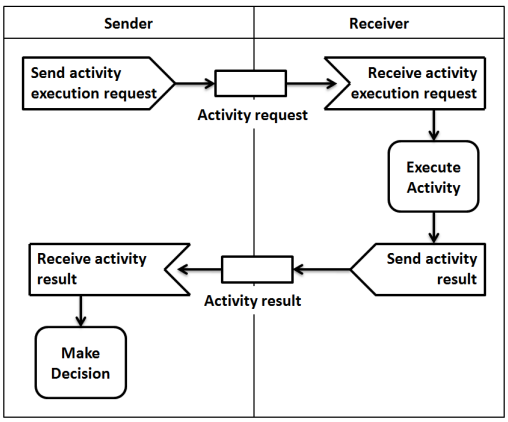
WAP4: Bidirectional Performative



WAP5: Notification



WAP6: Information Request



WAP7: Decision

Figure 1 Simplified versions of the seven WAPs defined in (Thom et al., 2009)

3 Semantics of Model Elements

Figure 3 shows an example of a travel booking process that has been modelled 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, which 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 patterns are present in this process. Note that Figure 3 already includes an indication of the patterns that were found manually by an analyst. Our goal is to discover these patterns automatically, provided that the model elements have been appropriately annotated.

3.1 Defining the WAP Ontology

In order to reason about concrete examples such as the one depicted in Figure 3, we need an *ontology* that provides a description of the patterns to be discovered, and we need to *annotate* the elements in the given process with the concepts defined in that ontology. For example, one should understand that the shape "Send request for booking" in Figure 3 is in effect a send signal with an activity request message as in WAP4; one should also realise that "Authorize trip" corresponds to a "Perform approval" activity as in WAP1; and so on. In order to do this, we introduce an ontology to specify these pattern elements.

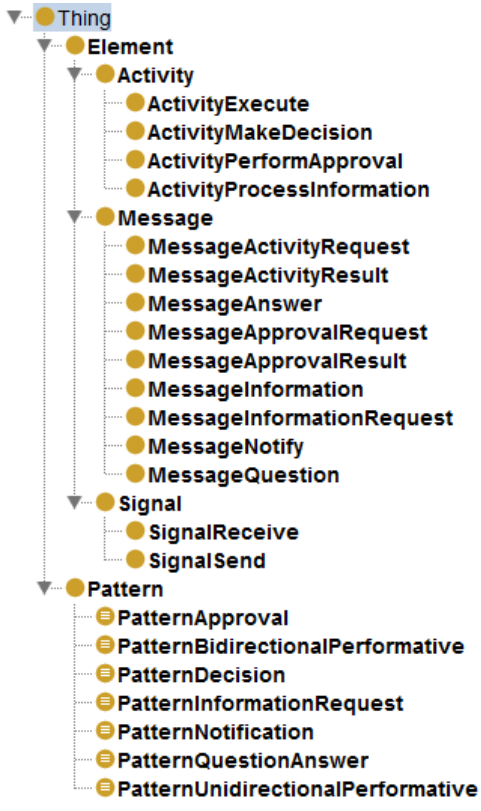


Figure 2 Class hierarchy for the WAP ontology.

Figure 2 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 WAPs. 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, which specifies the elements that the pattern contains. For example, the definition for WAP1 can be written as follows:

PatternApproval ≡

```
Pattern
and hasElement some (SignalSend
                     and hasMessage
                         some MessageApprovalRequest)
and hasElement some (SignalReceive
                     and hasMessage
                         some MessageApprovalRequest)
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 the elements of 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.

We should mention that this is not the first time that an ontology for workflow activity patterns has been devised. In (Thom et al., 2008) the authors make use of a WAP ontology for the purpose of supporting process modelling; in this case the ontology describes the patterns and the relationships between them in order

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

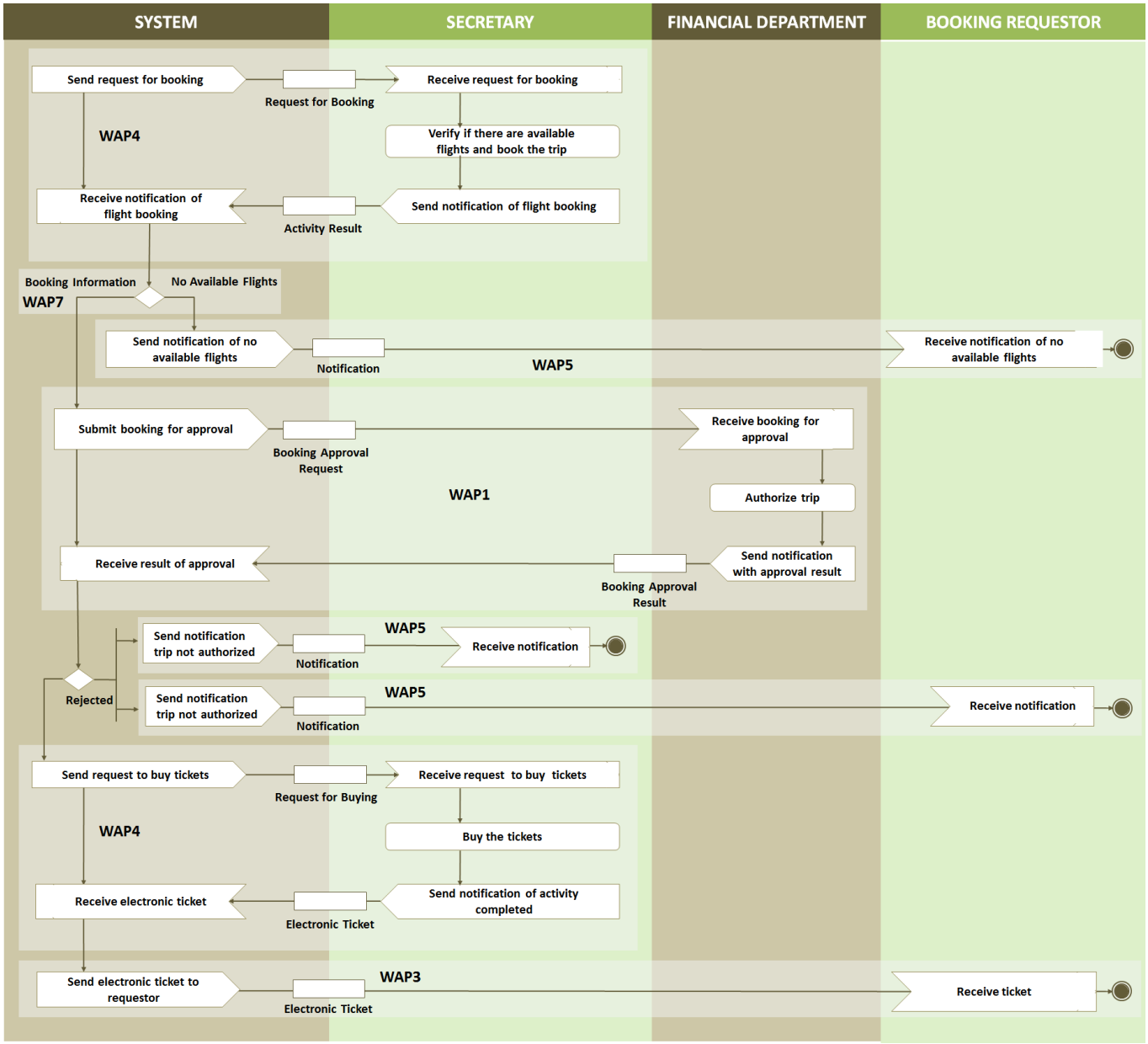


Figure 3 Travel booking example, adapted from (Thom et al., 2009)

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 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 elements of these patterns, and on how these patterns are defined in terms of the elements they contain.

3.2 Semantic Annotation of Model Elements

While the ontology above defines the classes, a given process model will contain the elements that can be seen as *individuals* of those classes. For example, the first shape "Send request for booking" in Figure 3 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 shapes in the model have been annotated with the corresponding classes from the ontology, creating these individuals is straightforward and can be done automatically. Then a reasoner can be invoked to infer the patterns that the process contains.

The critical point here is precisely in correctly annotating the elements, e.g. knowing that "Send request for booking" corresponds to two classes (SignalSend and MessageActivityRequest) and "Authorize trip" corresponds to ActivityPerformApproval. This annotation must be done manually by the analyst; in terms of effort, it is similar to other approaches that involve semantic annotation of business processes, e.g. (Born et al., 2007; Zouggar et al., 2008; Filipowska et al., 2009). Still, the annotation is made difficult by the fact that the shapes in a process model make use of a domain-specific vocabulary and are often labelled in different ways. To facilitate this task, it would be desirable to have the shapes in a process model labelled in a consistent way, such as using *verb-object* style, as recommended by (Mendling et al., 2010).

For the process in Figure 3, we would have:

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 annotation, the individuals and their properties can be generated automatically. For each class in the annotation, 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 being a member of several classes, which means that one can find in that 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

From the WAP ontology and the individuals above, a semantic reasoner is 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, a 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 (e.g. Process1) 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:

```
Process1 rdfs:subClassOf PatternApproval
Process1 rdfs:subClassOf PatternBidirectionalPerformative
Process1 rdfs:subClassOf 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 3, 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, and did not include it in the annotation.
- With regard to WAP3, that part of the process in Figure 3 is inferred as an instance of WAP5 rather than WAP3. This is because the message has been annotated as MessageNotify. However, it appears that the analyst originally thought that it was a MessageActivityRequest.

4 Capturing the Sequence of Events

In the previous section we have shown how it is possible to discover, through the use of reasoning, the presence WAPs in a process model, provided that a suitable annotation of the model elements is available. In this section we address the problem of checking not only that the pattern elements are present, but also that they are present in the correct order, as specified in the original WAPs of Figure 1.

For this purpose, and rather than analysing the control-flow of the process model, we resort to the sequence of events recorded during process execution. The use of event logs is very common in the area of process mining (van der Aalst et al., 2003b), for example to discover control-flow models (van der Aalst et al., 2004) and social networks (van der Aalst et al., 2005), or to study conformance (Rozinat and van der Aalst, 2008), among other issues. Here we use the event log as a means to retrieve the sequential behaviour of the process.

4.1 Translating the Event Log

An event log is a list of recorded events, where each event typically contains a reference to an activity (task id) that has been performed, the process instance (case id) that the activity belongs to, the user (originator) who performed the activity, and the time and date (timestamp) of when the activity was completed (van der Aalst et al., 2007). Such event log is usually retrieved from the execution of a business process on a workflow system, but it can also be obtained by simulation of the given process (Medeiros and Günther, 2005).

For our purpose, the event log needs to contain only the case id and task id, in chronological order. The sequence of tasks executed within a process instance, i.e. the sequence of task-ids associated with the same case-id, is called a *trace* (van der Aalst et al., 2004). A sample trace of the process in Figure 3 is as follows:

```
Send request for booking
Receive request for booking
Verify if there are available flights and book the trip
Send notification of flight booking
Receive notification of flight booking
Send notification of no available flights
Receive notification of no available flights
```

Provided with the mapping of Section 3.2, this trace can be translated into the following sequence:

```
SignalSend MessageActivityRequest
SignalReceive MessageActivityRequest
ActivityExecute
SignalSend MessageActivityResult
SignalReceive MessageActivityResult
SignalSend MessageNotify
SignalReceive MessageNotify
```

Since the control-flow in Figure 3 admits only 3 possible paths, there are a limited number of possible traces (the parallel execution of the branches beginning with "Send notification trip not authorized" may generate some additional traces). These traces can be translated in a similar way as above, and this translation can be done automatically for all traces based on the provided annotation of the model elements.

For the trace above, the approach described in the previous section would detect the presence of WAP4 and

WAP5 regardless of the order in which the elements appear in the trace. However, these patterns should only be detected if their elements appear in the correct order. For this purpose, we introduce the notion of an order relation, to be included in the WAP ontology.

4.2 Adding an Order Relation

In the trace above, the events that belong to the same WAP were recorded not only in sequence, but also consecutively in the event log. In practice, this may not be the case, as the events may become interspersed with other activities, namely those originated by parallel branches. The order relation for pattern elements should therefore make use of the notion of weak (rather than strict) order. The notion of weak (vs. strict) order has already been used extensively to capture the behavioural profiles of business processes from the sequence of events recorded in an event log (Weidlich et al., 2011).

In the present context, the definition of weak order applies to any pair of activities that follow one another, not necessarily in consecutive order. Formally, two activities a_i and a_j in a trace $\sigma = \langle a_1, \ldots, a_n \rangle$ are in weak order if and only if $1 \leq i < j \leq n$. To express this relation, we introduce the object property followedBy with domain Element and range Element, and we redefine the class expressions for the WAPs using this property. The definition for WAP1 becomes:

```
PatternApproval ≡

Pattern
and hasElement some
  (SignalSend
   and hasMessage some MessageApprovalRequest
   and followedBy some
     (SignalReceive
      and hasMessage some MessageApprovalRequest
      and followedBy some
        (ActivityPerformApproval
         and followedBy some
           (SignalSend
            and hasMessage some MessageApprovalResult
            and followedBy some
              (SignalReceive
               and hasMessage some MessageApprovalResult
              )
           )
        )
     )
  )
```

The definitions for the remaining patterns are analogous. It should be noted that the property followedBy is transitive, so that if A followedBy B and B followedBy C hold, then it can be inferred that A followedBy C holds as well. Such transitivity facilitates the specification of the order relations in the event log, since it is necessary to specify only the order relation between consecutive events for the reasoner to infer the order relation between all pairs of events.

4.3 Creating the Individuals

As before, in Section 3.3, the individuals can be created automatically from the event log. For example, the trace in Section 4.1 will result in the following individuals being created:

```
Element01 : SignalSend
Element02 : MessageApprovalRequest
Element01 hasMessage Element02

Element03 : SignalReceive
Element04 : MessageApprovalRequest
Element03 hasMessage Element04

Element05 : ActivityPerformApproval

Element06 : SignalSend
Element07 : MessageApprovalResult
Element06 hasMessage Element07

Element08 : SignalReceive
Element09 : MessageApprovalResult
Element08 hasMessage Element09

Element10 : SignalSend
Element11 : MessageNotify
Element10 hasMessage Element11

Element12 : SignalReceive
Element13 : MessageNotify
Element12 hasMessage Element13

Trace1 : Pattern
Trace1 hasElement Element01
Trace1 hasElement Element03
Trace1 hasElement Element05
Trace1 hasElement Element06
Trace1 hasElement Element08
Trace1 hasElement Element010
Trace1 hasElement Element012

Element01 followedBy Element03
Element03 followedBy Element05
Element05 followedBy Element06
Element06 followedBy Element08
Element08 followedBy Element10
Element10 followedBy Element12
```

Due to the transitivity of the property followedBy, all order relations between non-consecutive elements will be inferred. Also, from these statements and the new class expressions, as defined in Section 4.2, the reasoner is able to infer:

```
Trace1 rdfs:subClassOf PatternBidirectionalPerformative
Trace1 rdfs:subClassOf PatternNotification
```

Similar results can be obtained for other traces with additional patterns. The SPARQL query of Section 3.4 needs no modification other than replacing Process1 by Trace1. Now, however, changing the order of events in the event log leads to different order relations, and therefore

different assertions involving the property followedBy. A pattern will only appear in the result if the corresponding elements follow each other in the same order as defined in the class expression.

4.4 Implementation

The WAP ontology was developed and tested in Protégé together with the Pellet Reasoner Plug-in[2]. Using Java code, we load the ontology and create the individuals with the Jena framework[3]. The Pellet reasoner[4] 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 (a Java object implementing the OntModel interface). Then we read a text file containing the annotation of the process elements. For each class in the annotation, we retrieve a class reference (OntClass) from the ontology model, and create an individual of that class by invoking OntClass.createIndividual(). The relations between individuals are established by retrieving references to the appropriate object properties. Finally, 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.

5 Case Study: A Purchase Process

The following case study is based on the purchase process of a Portuguese company, implemented in a commercial workflow system (BizAgi[5]). The process is structured according to the following main stages:

- An employee fills out a requisition form for a certain product and sends it to the warehouse, which checks how many units are available in stock and returns the result.
- After receiving the result from the warehouse, the employee checks if there are enough units and, in that case, asks the warehouse to dispatch the product, receiving a confirmation in the end.
- However, in case the stock level is insufficient, the employee creates and submits a purchase request for approval by a manager. If approval is not granted, the process ends immediately. Otherwise, the process continues with the employee relaying a purchase order to the purchase department.
- The purchase department orders the product from a supplier and makes the warehouse aware of the incoming merchandise. Once the warehouse receives the product, it notifies the employee and forwards the invoice to the purchase department.
- The purchase department takes care of paying to the supplier and, once this is finished, notifies the employee that the requisition can be closed.
- Upon receiving note that payment is done, the employee closes the requisition and notifies the manager that the purchase request that had been previously approved is now complete.

Detalhes	De	Usuário	Data
Detalhes	Preencher requisição	joao	terça-feira, 11 de outubro de 2011
Detalhes	Enviar pedido de nível de stock	joao	terça-feira, 11 de outubro de 2011
Detalhes	Receber pedido de nível de stock	miguel	terça-feira, 11 de outubro de 2011
Detalhes	Consultar stock	miguel	terça-feira, 11 de outubro de 2011
Detalhes	Enviar resultado do stock	miguel	terça-feira, 11 de outubro de 2011
Detalhes	Receber resultado do stock	joao	terça-feira, 11 de outubro de 2011
Detalhes	existe stock?	joao	terça-feira, 11 de outubro de 2011
Detalhes	Enviar pedido de despacho do produto	joao	terça-feira, 11 de outubro de 2011
Detalhes	Receber pedido de despacho do produto	miguel	terça-feira, 11 de outubro de 2011
Detalhes	Despachar produto	miguel	terça-feira, 11 de outubro de 2011
Detalhes	Enviar resposta de despacho concluído	miguel	terça-feira, 11 de outubro de 2011
Detalhes	Receber resposta de despacho concluído	joao	terça-feira, 11 de outubro de 2011

Figure 4 Event log for a trace of the purchase process

For the purpose of this case study, the process has been augmented with additional activities in order to represent send and receive signals. The inclusion of these activities was necessary because the system event log does not record the exchange of messages between participants, only that some participant has performed some task. To make it easier to demonstrate the proposed approach, we modelled these exchanges explicitly. The send and receive signals are included as extra, "do-nothing" activities that do not affect the overall flow of the process, except for the fact that each user will visualise the same form for a second time, either after completing the current task (when sending the output to others) or before completing the current task (when receiving the input from others).

Figure 5 and Figure 6 show the overall structure of the purchase process. Since the activity labels are in Portuguese, we will not delve further into it, and instead we just note that the first lane is the employee, the second one is the warehouse, the third one represents the manager, and the fourth stands for the purchase department.

During this study we were able to collect two kinds of traces: one where the desired product was available in stock and was therefore dispatched immediately from the warehouse; and another where the product was not in stock and a purchase request was submitted, approved, and processed to the end (there were no instances of purchase requests that were not approved). Figure 4 shows a screenshot for a trace of the first kind, as recorded in the event log.

Based on the process model, we established a mapping between the activity labels and the classes in the WAP ontology, which is effectively equivalent to a semantic annotation of those activities. Using such

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

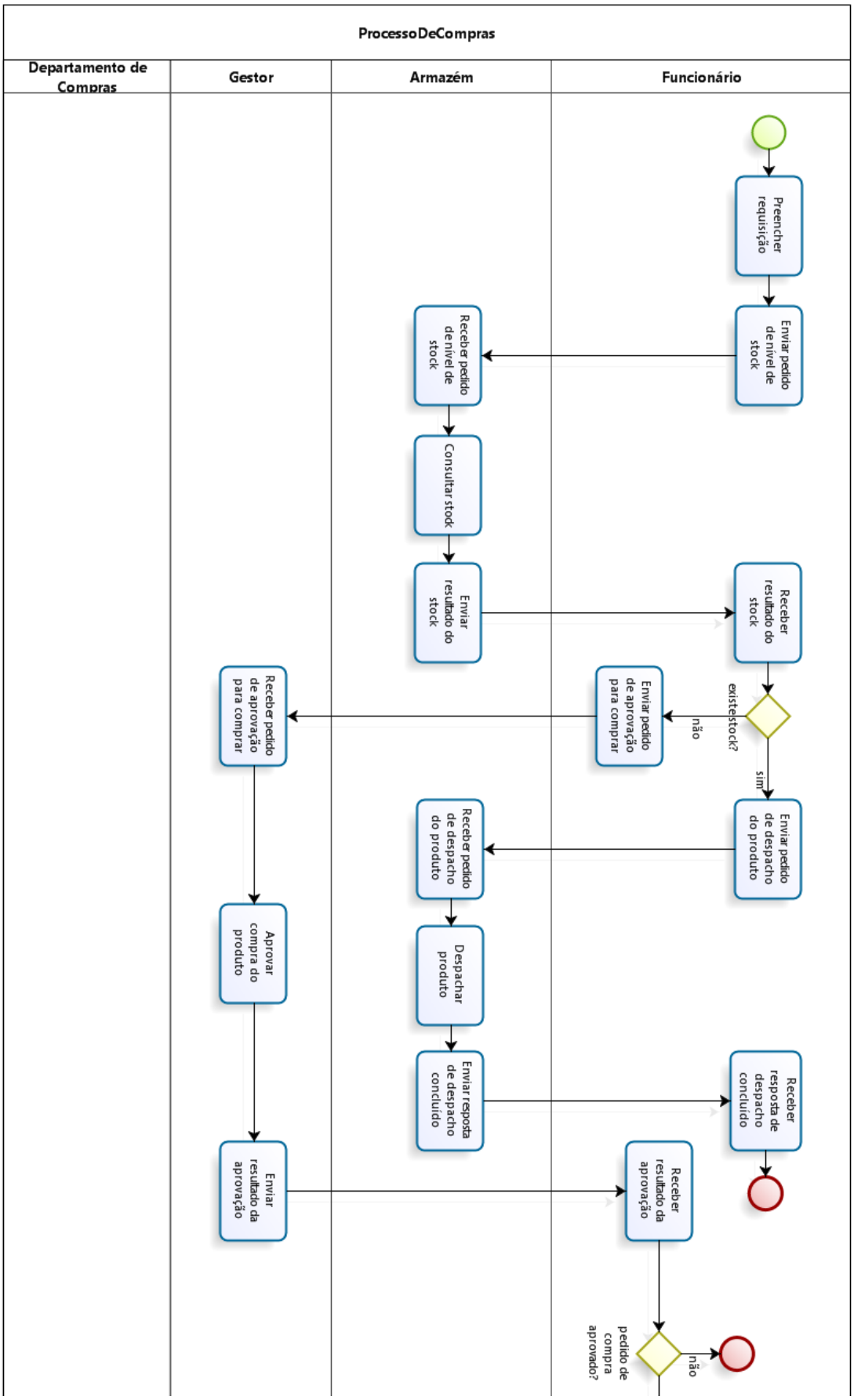


Figure 5 Purchase process as implemented in BizAgi (part 1 of 2)

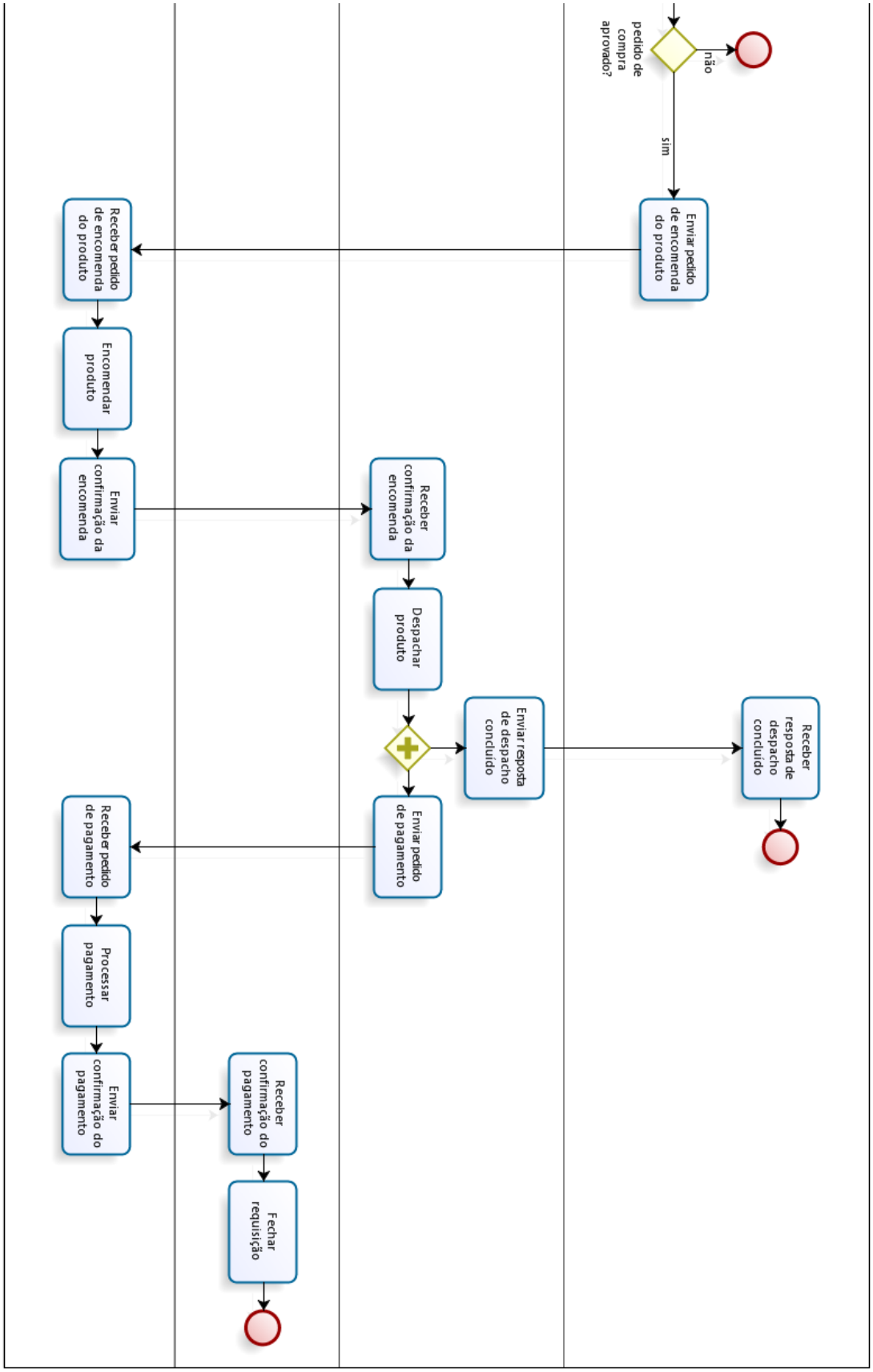


Figure 6 Purchase process as implemented in BizAgi (part 2 of 2)

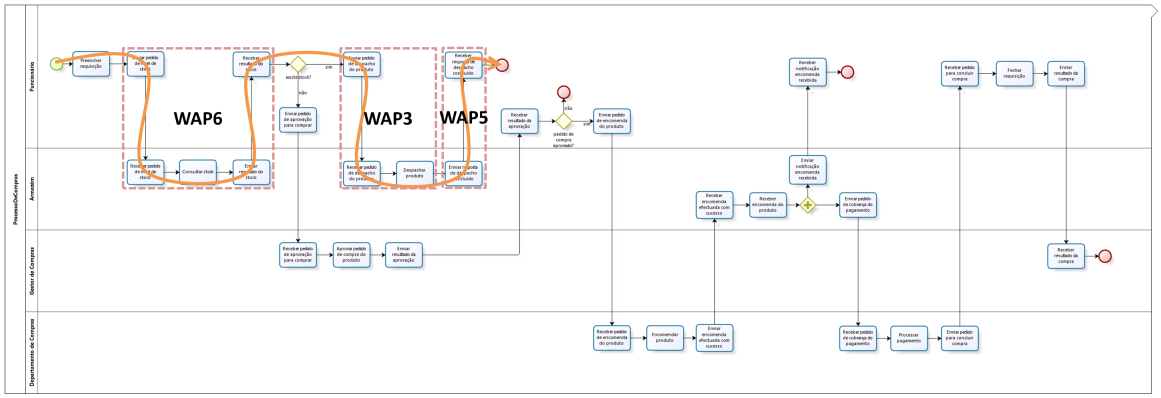


Figure 7 WAPs discovered in Trace1 of the purchase process

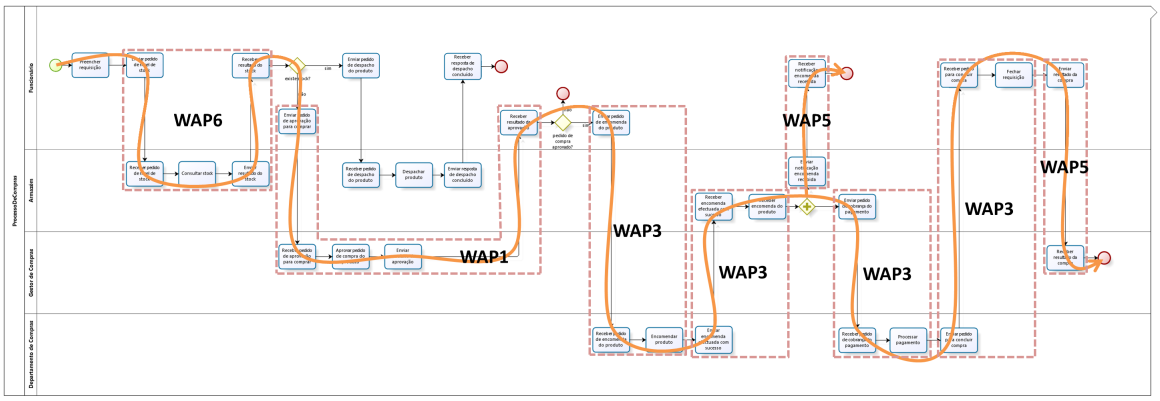


Figure 8 WAPs discovered in Trace2 of the purchase process

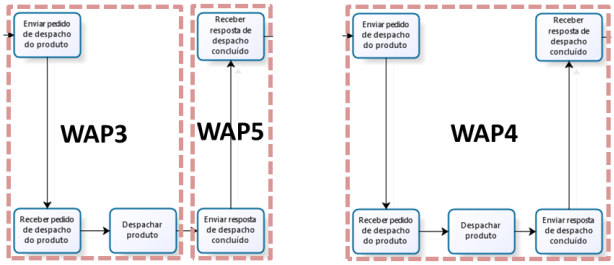
annotation, we translated the event logs mechanically, by replacing each event by its corresponding class or classes, as described in Section 4.1. Since only two kinds of traces were observed in the event log, we picked up one instance of each, which we denoted by Trace1 and Trace2. Figure 7 and Figure 8 highlight the path of each trace in the process model, and show the WAPs discovered in each of those traces.

It should be noted that a different annotation of the model elements may lead to the discovery of different WAPs. For example, Figure 9 illustrates the effect of a different annotation for the part of the process where the warehouse receives a request to dispatch the product and then returns a confirmation that the product has been dispatched. If this confirmation is taken as a notification, then the annotation becomes:

SignalSend MessageActivityRequest
SignalReceive MessageActivityRequest
ActivityExecute
SignalSend MessageNotification
SignalReceive MessageNotification

The annotation above leads to the discovery of WAP3 and WAP5, as in the original trace depicted in Figure 7. The first three events become a unidirectional performative (WAP3) and the last two events are a notification pattern (WAP5). On the other hand, if the confirmation from the warehouse is interpreted as a result, the annotation becomes:

SignalSend MessageActivityRequest



(a) Original annotation (b) Alternative annotation

Figure 9 WAPs discovered with different annotations

SignalReceive MessageActivityRequest
ActivityExecute
SignalSend MessageActivityResult
SignalReceive MessageActivityResult

In this case, the five events fit into the definition of a bidirectional performative (WAP4), as shown in Figure 9(b). In practice, it is up to the analyst to decide which annotation is more correct. At the present stage, establishing the semantics of the activities in a process requires human interpretation, so different results may arise from the same process. In any case, the approach described here is able to discover the presence of WAPs in a way that is consistent with the provided annotation.

6 Conclusion

In this work we have introduced an approach to automate the discovery of WAPs in business processes by means of reasoning over an ontology. In this ontology, the classes define the elements that each pattern contains, and the individuals represent the elements of the given process. These individuals can be generated automatically from an event log recorded during process execution. Both the semantics and the sequence of events are used to determine whether a given pattern is present in the process.

In future work, we intend to focus on tool support for the semantic annotation of model elements, and in particular on facilitating the annotation of elements which have a mapping to ontology classes other than one-to-one, such as one-to-many or many-to-one. This would avoid the need to insert artificial activities in the process, as we did in the case study to represent signals. Another branch for future work is to support more elaborate versions of the same patterns, including the iterative and concurrent versions of certain WAPs.

Acknowledgment

We are grateful to the PNPD Program from the Brazilian Coordination for the Improvement of Graduated Students (CAPES).

References

Born, M., Dörr, F., and Weber, I. (2007). User-friendly semantic annotation in business process modeling. In *Web Information Systems Engineering WISE 2007 Workshops*, volume 4832 of *LNCS*, pages 260–271. Springer.

Dietz, J. L. (2006). The deep structure of business processes. *Communications of the ACM*, 49(5):58–64.

Filipowska, A., Kaczmarek, M., and Stein, S. (2009). Semantically annotated EPC within semantic business process management. In *Business Process Management Workshops*, volume 17 of *LNBIP*, pages 486–497. Springer.

Medeiros, A. K. A. D. and Günther, C. W. (2005). Process mining: Using CPN tools to create test logs for mining algorithms. In *Proceedings of the Sixth Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools*, pages 177–190.

Medina-Mora, R., Winograd, T., Flores, R., and Flores, F. (1992). The action workflow approach to workflow management technology. In *Proceedings of the 1992 ACM conference on Computer-Supported Cooperative Work*, CSCW '92, pages 281–288. ACM.

Mendling, J., Reijers, H., and Recker, J. (2010). Activity labeling in process modeling: Empirical insights and recommendations. *Information Systems*, 35(4):467–482.

Rozinat, A. and van der Aalst, W. M. P. (2008). Conformance checking of processes based on monitoring real behavior. *Information Systems*, 33(1):64–95.

ter Hofstede, A. and Dietz, J. (2003). Generic recurrent patterns in business processes. In Weske, M., editor, *Business Process Management*, volume 2678 of *LNCS*, pages 1018–1018. Springer Berlin / Heidelberg.

Thom, L. H., Reichert, M., Chiao, C., Iochpe, C., and Hess, G. (2008). Inventing less, reusing more, and adding intelligence to business process modeling. In *Database and Expert Systems Applications*, volume 5181 of *LNCS*, pages 837–850. Springer.

Thom, L. H., Reichert, M., and Iochpe, C. (2009). Activity patterns in process-aware information systems: basic concepts and empirical evidence. *International Journal of Business Process Integration and Management*, 4(2):93–110.

van der Aalst, W., ter Hofstede, A., and andA.P. Barros, B. K. (2003a). Workflow patterns. *Distributed and Parallel Databases*, 14(1):5–51.

van der Aalst, W., van Dongen, B. F., Herbst, J., Maruster, L., Schimm, G., and Weijters, A. J. M. M. (2003b). Workflow mining: A survey of issues and approaches. *Data and Knowledge Engineering*, 47(2):237–267.

van der Aalst, W., Weijters, T., and Maruster, L. (2004). Workflow mining: discovering process models from event logs. *IEEE Transactions on Knowledge and Data Engineering*, 16(9):1128–1142.

van der Aalst, W. M. P., Reijers, H. A., and Song, M. (2005). Discovering social networks from event logs. *Computer Supported Cooperative Work*, 14(6):549–593.

van der Aalst, W. M. P., Reijers, H. A., Weijters, A. J. M. M., van Dongen, B. F., Alves de Medeiros, A. K., Song, M., and Verbeek, H. M. W. (2007). Business process mining: An industrial application. *Information Systems*, 32(5):713–732.

Weidlich, M., Polyvyanyy, A., Desai, N., Mendling, J., and Weske, M. (2011). Process compliance analysis based on behavioural profiles. *Information Systems*, 36(7):1009–1025.

Zouggar, N., Vallespir, B., and Chen, D. (2008). Semantic enrichment of enterprise models by ontologies-based semantic annotations. In *Proceedings of the 12th International EDOC Conference Workshops*, pages 216–223. IEEE Computer Society.