Discovering Cross-Perspective Semantic Definitions from Process Execution Logs

Stefan Schönig, Christoph Günther, Michael Zeising, and Stefan Jablonski
University of Bayreuth
Chair of Applied Computer Science IV
Bayreuth, Germany
{stefan.schoenig, christoph.guenther, michael.zeising, stefan.jablonski}@uni-bayreuth.de

Abstract - **In this paper, we suggest a two-phase declarative process mining approach discovering explicit, cross-perspective semantic definitions. Cross-perspective semantic definitions are interesting and important for the analysis of business processes, because they reveal dependencies that are not obvious on the first look. They allow for a comprehensive examination of the recorded process execution information and enable the discovery of coherency between different process-involved entities and perspectives. Using the described cross-perspective semantic definitions, we additionally introduce an approach for simplifying less-structured process models.**

Keywords - Process mining; semantic definitions; business rules; ontology; semantic process modelling.

I. INTRODUCTION

Process modelling is an expensive and cumbersome task. Using process mining techniques, it is possible to discover process models automatically [10]. Moreover, event logs can be checked to assess conformance and compliance with respect to already defined processes [10]. Process mining has been applied in various domains ranging from healthcare and e-business to high-tech systems and auditing [5, 6]. However, many process mining techniques produce "spaghetti-like" models that tend to be large and complex, especially in flexible environments where process executions involve multiple alternatives [1, 2]. This "overload" of information is caused by the fact that traditional mining techniques construct imperative models explicitly encoding all possible behaviours [1, 10]. This sort of complexity arises when a huge number of execution paths exists (path complexity) [23]. When process models are becoming too complex, people cannot interpret them anymore and therefore cannot improve them. In order to face this problem, we leave the imperative world and focus on the generation of declarative process models. Declarative process modelling techniques reduce path complexity such that complex applications can be described by comprehendible process models.

In contrast to imperative modelling, declarative models concentrate on describing what has to be done and the exact step-by-step execution order is not directly prescribed. There are several process mining approaches like [1, 19, 20] that are discovering declarative process models. Here, the meaning of model elements is defined by explicit semantic definitions. Furthermore, several approaches [2, 3, 4, 12] filter the information contained in a log and to simplify less-structured imperative process models by discovering common execution patterns. However, the approaches named above have a common drawback: they are mainly examining the behavioural perspective, i.e., the control-flow. These methods are discovering semantic definitions considering the execution order of process steps without facing possible coherency with other perspectives. We think that especially this hidden coherency between perspectives should be outlined by discovery algorithms. That is why we suggest a mining approach, based upon user-defined cross-perspective semantic definitions. That means these semantic definitions spread over different entities and perspectives, e.g., a process execution order (behavioural perspective) could depend on the performing agents' position (organisational perspective). The semantic definitions are constituted through the analysis of the different perspectives recommended by the *perspective-oriented process modelling* (POPM) approach [15]. The user-defined assembly of semantic definitions allows analysts to shape the discovery process to extract the semantic definitions that are most important and interesting for them [1].

Cross-perspective semantics is especially interesting and important for the analysis of business processes, because it reveals dependencies that are not obvious on the first look. It allows for a comprehensive examination of the recorded process execution information and enables the discovery of coherency between different process-involved entities and perspectives. Beyond, our approach discovers semantic definitions that can be based upon properties of an ontology, containing further information about process-involved entities or participants. That means that the semantic definitions to be searched can partly consist of properties that have to be extracted from an underlying ontology. This functionality is similar to [13, 14], however we apply this possibility in the context of process discovery instead of conformance checking. Using the described cross-perspective semantic definitions, we additionally introduce an approach for simplifying less-structured process models.

This paper is organized as follows: Section II introduces the fundamental assembly and the two phases of the approach. In Sections III and IV, these two phases are described in detail. In Section V, related work is discussed. The paper is finally concluded in Section VI.

II. DISCOVERING CROSS-PERSPECTIVE SEMANTIC DEFINITIONS FROM PROCESS EXECUTION LOGS

Information systems typically log various kinds of information about process execution. The starting point for

 ISBN: 978-1-61208-223-3

process mining is an event log. An event log consists of a set of traces whereat each trace is a sequence of events corresponding to a particular case, i.e., one process instance [7]. Each events record refers to a single process step and typically has a timestamp. These facts also form the preliminaries for our approach. We assume an existing event log recording different perspectives of process execution. Table I shows a fragment of such a process execution log. While the PID-column assigns each event to unique process identifier, the case-column assigns each event to a single process case, i.e., a single process instance. Furthermore, the action type of each event is recorded. The following columns are an example for further information that should be logged during process execution in order to be able to discover cross-perspective business rules. We recommend to record data based upon the different aspects of the *perspective-oriented process modelling* (POPM) [15]:

Functional perspective: the functional perspective identifies a process step and defines its purpose. Also the composition of a process is determined by this perspective. Hence, the log should contain a common process identifier the corresponding event can be linked to.

Data perspective: the data (flow) perspective defines data used in a process and the flow of data between process steps. Therefore, the log should record documents or generally information that was used by the current process step as well as the data that was produced.

Operational perspective: the operational perspective specifies which operation (service) is invoked in order to execute a process step. It relates processes to services stemming from (external) service libraries. Here, the log should contain tools, applications or services that were used during performing the currently executed process step.

Organisational perspective: the organisational perspective defines agents (for instance users, roles) who are eligible and/or responsible to perform a process step. Therefore, the log contains information about the process executor. The personal information is enriched by group and role memberships.

Behavioural perspective: the behavioural perspective is used to define causal dependencies between process steps (e.g. step B may only be executed after step A). Often these dependencies are called control flow. The information in the log concerning this perspective is formed by the recorded timestamp of each event.

TABLE I. A FRAGMENT OF A PROCESS EXECUTION LOG.

Event	PID	Case	Action	Agents	Data	Tools	Time
1	A	**1**	Start	Head	Doc 1	Word	…
2	A	**1**	Finish	Head	Doc 1	Word	
3	D	*2*	Start	Agent 3	Doc 3	Excel	
4	D	*2*	Finish	Agent 3	Doc 3	Excel	
5	B	**1**	Start	Agent 2	Doc 2	Word	
6	C	*2*	Start	Agent 3	Doc 2	Excel	
7	B	**1**	Finish	Agent 2	Doc 2	Word	
8	C	2	Finish	Agent 3	Doc 2	Excel	
9	C	**1**	Start	Trainee	Doc 3	Word	
10	A	2	Start	Head	Doc 1	Word	
11	C	**1**	Finish	Trainee	Doc 3	Word	
12	D	**1**	Start	Trainee	Doc 4	Word	
13	D	**1**	Finish	Trainee	Doc 4	Word	
…							

The existence of an event log of such a shape allows for the comprehensive examination of various perspectives within one approach. Therefore, we propose a two phase approach to analyse a process execution log.

Phase-1 (Pre-processing the log to instance graphs; Section IV). Here, the event log is generally analysed and transformed into various graph data structures that allow for the flexible search of user-defined semantic definitions.

Phase-2 (Discovery of cross-perspective semantic definitions; Section V). This phase discovers cross-perspective semantic definitions concerning only one process as well as relations between two processes. Semantic definitions are used to encapsulate processes.

III. PRE-PROCESSING THE LOG TO INSTANCE GRAPHS

In this section, we focus on the construction of so-called *instance graphs*. An instance graph describes the execution order of process steps of a process (case), i.e. one single execution path of a process. For our particular cross-perspective purpose, we feature the graphs of [24, 25] with context data of the organisational, data and the operational perspective. Instance graphs also show parallelism if parallel (independent) branches have been executed. An instance graph consists of a set of nodes N and a set of edges E. Every node $n \in N$ has the following fields: process name, performing agent, used document and used tool support. Every edge $e \in E$ has two fields describing how two processes are connected: execution type (parallel or sequence) and distance (direct or transitive). Every instance graph is a complete graph. First, we separate the recorded events according to their corresponding case/instance id.

Therefore, we assemble a list for each case represented in the log and assign the events according to their case ids. With the help of these lists, we can now classify the relation between two (sub-)processes within one process case. The classification is based upon the event types of two succeeding events. Here, we make the same assumptions as [24, 25]. As already mentioned, we distinguish between parallel execution and direct sequential execution. Consider two processes *A* and *B*. We deduce that two processes are executed in parallel if process *A* is started before process *B* is started and completed before *B* is completed but after the start of *B*. This would result in the event sequence: *Start A*, *Start B*, *Finish A*, *Finish B* (Fig.1 ①). Furthermore, the two processes are also executed in parallel if process *A* is started before process *B* is started and completed after process *B* is completed. The resulting event sequence would look like this: *Start A*, *Start B*, *Finish B*, *Finish A* (Fig.1 ②). In addition to parallel execution, we mark direct sequential execution. Two processes *A* and *B* are executed in a direct sequence if process *B* is started directly after process *A* has been completed (we say "*B* is started after *A* finished"). The

resulting event sequence therefore is: *Start A*, *Finish A*, *Start B*, *Finish B* (Fig. 1 ③).

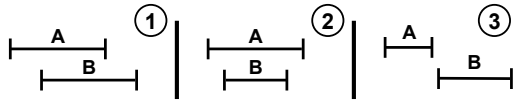

Figure 1. Classification of behavioural relations of processes.

On the basis of the classification above, an instance graph of an event list is created as follows. We generate a graph by running through the case-specific event list. For every newly occurring process *A* within the list, we create a new node *A* within a graph and assign the corresponding process context to the fields of the node, i.e., recorded agent, documents as well as tools. For every direct sequential-relation of two processes *A* and *B*, we add an edge of execution type "sequence" and distance "direct" between the node representations of *A* and *B* within the graph (formal: $A \rightarrow B$). For every parallel execution between two processes *A* and *B*, we add an edge of execution type "parallel" and distance "direct" between the node representations of *A* and *B* within the graph ($A \leftrightarrow B$). Finally, the existing graph is extended by edges that have been generated by the transitive closure of the graph. If the graph already contains two edges $A \rightarrow B$ and $B \rightarrow C$, we add an edge of execution type "sequence" and distance "transitive" between the node representations of the processes A and C ($A \gg C$). Note that in general, it is not possible to infer $A \leftrightarrow C$ from $A \leftrightarrow B$ and $B \leftrightarrow C$. Fig. 2 shows three different instance graphs of a process based upon the log fragment of Table 1 (for space reasons, the table just shows the activities of two instances). Considering graph 1, case 1 had the execution trace *A*, *B*, *C*, *D*, containing only direct sequential-relations and no parallelism. Exemplarily, graph 1 additionally contains the information that the agent "Head of Department" executed process *A* by using *Document 1* supported by *MS Word*. The two other graphs can be interpreted in the same way.

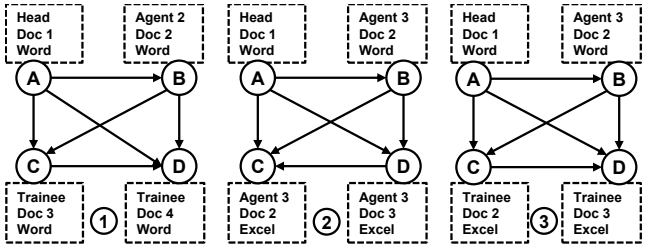

Figure 2. Instance graphs based on three different process cases.

IV. Discovery of Cross-Perspective Semantics

A. Classification of Semantics

Meta-modelling frameworks like the *Open-Meta-Modelling Environment* [16] offer the possibility to feature and adapt process modelling languages with a variety of user-defined (domain-specific) modelling elements. This requires a clear specification of the meaning of modelling constructs in order to avoid misunderstanding between modeller and programmer. This can be achieved by explicit semantic definitions of model elements. Semantic definitions are already used to validate executed processes in the context of conformance checking [10]. Hence, logs can be validated with the help of logical reasoners. In this paper, we scan event logs in order to discover semantic definitions. First, we introduce the assembly and representation of semantic definitions. Here, we use the *Semantic Web Rule Language* (SWRL) [21] to define semantics of modelling elements. Every semantic definition, e.g., SWRL rule, has a left and a right side. The left side contains the conditions that have to be satisfied so as to infer the consequences on the right side. Furthermore, the left side contains an indicator for the corresponding modelling construct whose semantics is defined. Conditions and consequences consist of atoms and assigned variables. The formalisation of such rules looks as follows:

$$Modeling\_Element\_Indicator \wedge Conditions \Rightarrow Consequences$$

The running example for this section consists of a domain-specific modelling element called *TraineeConnector*. This semantics expresses that any two processes *P1* and *P2* connected by this semantics have a strict execution order in case that a trainee employee performs the processes. Here, the cross-perspective nature becomes obvious: the behavioural perspective depends on the organisational perspective. The representation of this semantics as an SWRL definition looks as follows:

$$\begin{aligned}&\text{TraineeConnector}(?C) \wedge \text{from}(?C, ?P_1) \wedge \text{to}(?C, ?P_2) \wedge \\ &\text{hasAgent}(?P_1, ?A) \wedge \text{hasAgent}(?P_2, ?A) \wedge \text{Trainee}(?A) \\ &\Rightarrow \text{startedAfterFinished}(?P_2, ?P_1)\end{aligned}$$

Note, that the algorithm also is able to discover established (single perspective) declarative semantic definitions. Exemplarily, the semantic definition of a "standard" sequential execution order in any case (a name for such semantics could be *StrictOrderConnector*) would look like this:

$$\begin{aligned}&\text{StrictOrderConnector}(?C) \wedge \text{from}(?C, ?P_1) \wedge \text{to}(?C, ?P_2) \\ &\Rightarrow \text{startedAfterFinished}(?P_2, ?P_1)\end{aligned}$$

The assembly of the core algorithm demands for the definition of a few terms describing the type of semantic definitions.

Attribute: Attributes describe modelling elements that represent semantics concerning a single process. An attribute does not express a relation to another process. An example would be the fact that a specific process has always to be executed by the head of department.

Connector: Connectors describe modelling elements that represent relations, e.g., connectors, between two processes. An example for a connector is the *TraineeConnector* from above.

Group: Group semantic defintions typically assign functional entities (e.g., atomic processes) to other functional entities (e.g., complex processes or pools).

 ISBN: 978-1-61208-223-3

There may also exist other semantic definitions, however, in this paper we focus on discovery of semantic definitions defined above with respect to comprehension reasons.

B. Discovery of Semantic Definitions

The validation of given semantic definitions represents the core functionality of the approach. The general approach is as follows:

1. Search for explicit semantic definitions (attributes as well as connector semantics) using the instance graphs.
2. Generation of the semantic graph, that contains attributes of single processes as well as connectors between processes.
3. Encapsulation of processes and generation of process hierarchies examining group semantic definitions.

The main principle of the validation procedure is described by means of connector semantics. Therefore, every instance graph has to be analysed. There can be a relation between every combination of two processes (x, y) for every semantic definition. The core algorithm is build-up as follows: At first, we assume that a relation exists between the currently observed processes x and y. Now, the left side of the semantic definition is examined. Therefore, the processes and the corresponding process context information have to satisfy the conditions. If the information within the instance graphs is not sufficient, additional data is extracted from an ontology. If all conditions are satisfied, the consequences are examined. We make use of the principal proof by contradiction. All elements of the right side of the definitions are negated. If the negation of the consequence could be found within an instance graph, it means the algorithm disproved the current semantics for the processes x and y. If we could not find any counter-example within the instance graphs, the semantics is valid for the processes x and y. Note, that our assumption only holds, if the log is scanned completely every time. The extracted semantics is only valid for the currently analysed knowledge base. The processes x and y are connected with the corresponding model element, whose semantics is defined by the rule. The fact that we use a proof by contradiction is based upon the closed-world assumption. If we cannot find a counter-example within any instance graph (these in combination with an underlying ontology reflects our knowledge base), the semantics is declared as valid for the currently considered processes. The formalisation of the described proof is as follows:

Discovery of Semantics s between processes P_1, P_2:

$proof = \text{true}$
∀ Instance Graphs
{
  IF $s.Conditions$ **THEN**
  {
    IF $\neg s.Consequences$ **THEN**
      $proof = \text{false}$
  }
}

IF $proof$ **THEN**
  Semantic s is discovered between P_1 and P_2

The assembly of a validation algorithm for attribute-semantic definitions is similar to the one above. However, the algorithm takes only one process x and in case of a valid semantics, an attribute is assigned to x instead of a connector. *Example:* Consider the three instance graphs of Fig. 2 as the knowledge base for this example. We now apply the algorithm to discover the *TraineeConnector* between the processes *C* and *D*. At first, graph 1 (Fig. 2 ①) is examined. In this case all conditions are satisfied as the processes C and *D* have assigned agents and the performing agents are obviously members of a class "Trainee". Therefore, the algorithm demands for the examination of the consequences. This is, in the case of the *TraineeConnector,* the fact that process *D* has to be started after the completion of process *C*. As the algorithm follows the principle proof of contradiction, we have to examine if the graph contains the contrary. That is obviously not the case as *D* is performed after *C* and therefore the *proof* variable stays true. In the next step, graph 2 (Fig. 2 ②) is examined. Here, the conditions are not satisfied, as the performing agent "Agent 3" is not member of a class "Trainee". Hence, the consequence must not be examined and the *proof* variable stays true. Note, that in this case *C* was performed after *D* (wrong order). However, this is not relevant as the performing agent is not a trainee. In the last instance graph (Fig. 2 ③) the situation is identical to graph 1. This is why the *proof* variable stayed true during the whole examination and the algorithm discovered the *TraineeConnector* between the processes *C* and *D*.

C. Generation of the Semantic Graph

During the validation of semantics, we assemble a new graph (called *semantic graph*) containing all the extracted attributes and relations between, processes. The graph consists of a set of nodes N and a set of edges E. The nodes represent the processes, whereas the edges represent relations between these processes. Every node $n \in N$ has the following fields: process name, performing agents, used documents, used tools and furthermore a list containing the discovered attributes for this specific process. As described before, connector semantic definitions represent relations between two processes. Therefore, connector semantic definitions are depicted as edges $e \in E$ within the semantic graph. An edge e has two fields describing the connector between two processes: connector-type (i.e., *TraineeConnector*) and distance (direct, transitive). The proof of an attribute semantics adds an entry to the attributes-list of the node representation of the corresponding process. Furthermore, the proof of a connector semantics concerning the processes x and y adds an edge between the node representations of these two processes with the corresponding connector type. The semantic graph of Fig. 3 highlights three exemplary visualised semantic definitions extracted on the basis of the three instance graphs of Fig. 2. The algorithm discovered that process *C* and *D* have a strict execution order only in case that a trainee employee performs these processes. Note, that we assume a closed-world and cannot found a counter-example in any instance graph. That is why the semantic graph shows the

TraineeConnector (visualised as a dashed arrow) between *C* and *D*. Moreover, the graph shows various other semantic definitions, discovered by analysing the instance graphs of Fig. 2. Consider the processes *A* and *B*. Here, the algorithm discovered a strict execution order in any case (i.e., the *StrictOrderConnector* visualised by a continuous arrow), as B has always been executed after *A*. Furthermore, the algorithm discovered an attribute of process *A* revealing that *A* has always been executed by the head of department (that is why *A* is visualised with black filling).

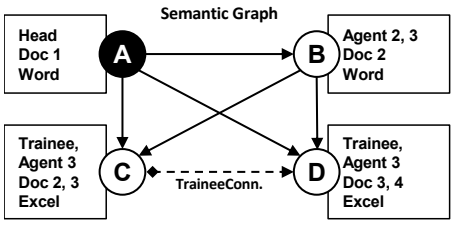

Figure 3. Semantic graph containing cross-perspective semantics.

Finally, the resulting graph contains all processes, the process context information and all attributes and connectors between processes that could have been extracted from a set of instance graphs.

D. Encapsulation of Processes

We propose to simplify and structure process models by examining group semantic definitions that additionally allow for encapsulating processes based on other perspectives. The approach allows for assigning processes to sets, i.e., groups, whose members offer common behaviour or characteristics with respect to a specific perspective. These groups of processes can be highlighted by assigning modelling elements to each predefined group semantics.

1) Encapsulation based on the behavioural perspective

In the first part of this section, we focus on encapsulation of processes on the basis of the behavioural perspective. Note that the execution order of processes could be based on various other perspectives. The execution order is for example mostly determined by the data flow. The encapsulation of processes based on the behavioural perspective has the aim to simplify process models by assigning atomic processes to complex processes as sub-processes. While an atomic process is associated with an executable activity, a complex process contains sub-processes [18]. This allows for the hierarchical composition of processes. Therefore, a complex process serves as a "*capsule*" for its sub-processes. The algorithm to encapsulate processes with respect to the behavioural perspective is based upon the following two rules:

$$\text{sameGroup}(?P_1, ?P_2) \wedge \text{startedAfterFinished}(?P, ?P_1) \Rightarrow \text{startedAfterFinished}(?P, ?P_2)$$

$$\text{sameGroup}(?P_1, ?P_2) \wedge \text{startedAfterFinished}(?P_1, ?P) \Rightarrow \text{startedAfterFinished}(?P_2, ?P)$$

The rules say: two processes P_1 and P_2 are in the same group (capsule) if every process P that is started (completed) after (before) P_1, is also started (completed) after (before) P_2. That means, two processes are in the same capsule, if they always have the same predecessors as well as the same successors with respect to all instance graphs. In order to discover this semantics, we need two rules as the control variable process P cannot be a predecessor and a successor at the same time. Therefore, we first have to check if P_1 and P_2 have the same predecessors and subsequently if they have the same successors. A formal definition of the encapsulation based on the behavioural perspective is as follows:

```
∀ process combinations (P1, P2)
∈ Set of processes recorded in the log
{
   proof = true
   ∀ Instance Graphs
   {
      P ∈ Set of processes recorded in the log
         ∀ p ϵ P \ {P1, P2}:
               IF s1.Conditions THEN
               {
                     IF ¬s1.Consequences THEN
                           proof = false
               }

               IF s2.Conditions THEN
               {
                     IF ¬s2.Consequences THEN
                           proof = false
               }
   }

   IF proof THEN
         P1, P2 ∈ Group x
}
```

It is obvious that the assembly of the algorithm is similar to the algorithm to discover connector semantics. However, here we have to examine two rules at the same time and a control variable *P* influences the examination.

Example: Consider again the three instance graphs of Fig. 2. We focus on the process combination *A*, *B*. In the first part of the formalism the focus is on common predecessors. Obviously, no instance graph contains predecessors for *A* and *B*, so the *proof* variable stays true. The second part of the algorithm examines the successors. Every process *P* that is started after the completion of *A*, has also to be started after the completion of *B*. Given that $P = \{C, D\}$ we have to examine which $p \in P$ are successors of *A* and check if p is also a successor of *B*. Focusing on the three instance graphs of Fig. 2 it is obvious that process *A* has the successors *C* and *D* in every case. Moreover, *C* and *D* are also successors of process *B* in every instance. Hence, the algorithm could not find a counter-example within the knowledge base and the *proof* variable stayed true during the examination. Finally, the algorithm assigned *A* and *B* to a set *Group x*. By applying this procedure recursively to the resulting sets, process hierarchies are discovered. In Fig. 4, we visualise

 ISBN: 978-1-61208-223-3

the group membership property by a box surrounding the contained processes.

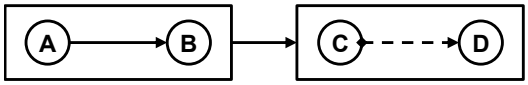

Figure 4. Exemplary visualisation of group semantics.

2) Encapsulation based on other perspectives

Besides, our semantic definitions allow for the encapsulation on the basis of other perspectives. In this paper, we exemplarily focus the encapsulation with respect to the organisational perspective. However, encapsulation based on any other perspective is possible. Exemplarily, we encapsulate processes based on the organizational perspective. If the agents' organization is not recorded in the log explicitly, we can extract it from ontologies by querying with the corresponding agent id. In our example, we visualise the group membership property again by a box element surrounding the contained processes. Note, that the resulting visualisation is related to pools in BPMN [17]. The group semantics to search for looks as follows:

$$\text{sameGroup}(?P_1, ?P_2) \wedge \text{hasAgent}(?P_1, ?A_1) \wedge \text{hasAgent}(?P_2, ?A_2) \wedge \text{employedAt}(?A_1, ?B1) \wedge \text{employedAt}(?A_2, ?B2) \Rightarrow sameAs(?B_1, ?B_2)$$

Here, we do not have to examine a common behaviour with respect to other processes and therefore there is no need for a control variable. In the case of the organisational perspective, we propose a solution to the challenge of cross-organisational mining. Consider a cross-organisational process, e.g., supply chains of manufacturing processes. Each involved organization records its activities to its specific event log. We propose to merge the different logs to one integrated log based on the recorded timestamps. The prerequisite in order to mine for instance graphs is that the involved organizations have a common instance ID. Note that especially in the case of a supply chain, the instance id could be based on a common product id. In the context of cross-organisational mining it could be interesting how the different parties interact with each other. Fig. 5 shows how cross-organisational interaction as for a product P_1 could be highlighted based on merging different logs and grouping different processes with respect to the performing agents' organization.

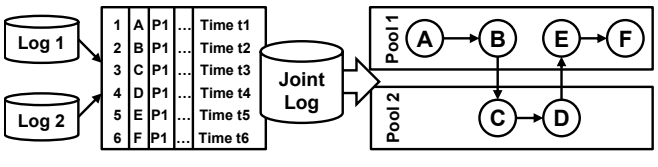

Figure 5. Merging of logs and encapsulation based on organizations.

V. Related Work

Starting point for process mining is a process execution log. The basis for analysis is pre-processing the available log. In this paper, the pre-processing is carried out by converting the log information to so-called instance graphs. These graphs are similar to the graphs of [11, 24, 25], however we feature them with further context data of process execution. There are already several algorithms and even complete tools, like the *ProM Framework* [8], that aim at discovering and generating process models automatically.

During the last decade, several algorithms have been developed, focusing different perspectives of process execution data. Van der Aalst et al. give a detailed introduction to the topic process mining and a recapitulation of research achievements in [5, 6, 10]. Many of these traditional process mining algorithms are imperative approaches. These methods construct imperative models explicitly showing all possible behaviours. Other ways to mine for process models are declarative approaches. Instead of explicitly specifying all the allowed sequences of events, declarative process models specify the possible ordering of events implicitly by constraints. Alongside with declarative approaches that are used for conformance checking of already existing models [10, 22], there are several declarative discovery algorithms like [1, 19, 20]. The major difference to these methods, that are mainly examining the behavioural perspective, is that our approach is based upon semantic definitions considering multiple perspectives of process data and includes the possibility to query ontologies. This enhances the declarative discovery process by functionality provided by semantic process mining [13, 14] and allows for revealing dependencies that are not obvious on the first look. In literature, several approaches [2, 3, 4, 12] are described to filter the information contained in a log and to simplify less-structured process models. However, these methods are limited to encapsulate processes based on common behaviour with respect to the control-flow. We propose to simplify and structure process models by examining cross-perspective semantic definitions.

VI. Conclusion and Outlook

In this paper, we suggested a declarative mining approach, based upon explicit, cross-perspective semantic definitions. Semantic definitions are constituted through the analysis of the different perspectives recommended by the *perspective-oriented process modelling* (POPM) approach and spread over different process-involved entities and perspectives. Cross-perspective semantic definitions are important for the analysis of business processes. Additionally, we proposed an approach to simplify and structure process models by examining cross-perspective semantics that allows for encapsulating processes based on various perspectives. By grouping processes with respect to the organisational perspective, we proposed a contribution to the challenge of cross-organisational mining. The *IEEE Task Force on Process Mining* stated the improvement of usability and understandability for non-experts as a challenge for future mining approaches [8, 9]. We strive for improving these issues with respect to the application environment as well as the presentation of results. The latter

 ISBN: 978-1-61208-223-3

is amended by the use of domain specific modelling elements, whereas the prospective integration in the *Open-Meta-Modelling Environment* [16] will allow for the simple definition of modelling elements with underlying semantics. The user should have the possibility to specify the semantics to search for and therefore to extend or reduce the scope of functions flexibly. A kind of instruction manual for process managers could look as follows: at first, analysts define the semantic definitions in SWRL notation they are most interested in. Next, a process modeller assigns user-defined model elements to these semantic definitions. The resulting model elements represent the domain specific process language that is used in the current case. After that, a log is selected as an input for the algorithm. Subsequently, the semantic process mining algorithm discovers the given semantic definitions from the language-catalogue. By traversing the resulting graph, it is possible to transform the information into a process model. Considering the assigned model elements, the process model based on the previously defined process meta-model, i.e., the language catalogue, is visualised. The discovered model can be discussed, possibly remodelled and finally be executed by a workflow engine.

We strive for an extensive application of our approach, including a detailed evaluation. Our future research activity in the field of process mining will face the problem that process events are typically not recorded in a unified manner. Therefore, we are developing an approach to align process logs recorded with different granularities.

REFERENCES

[1] Maggi, F.M., Mooij, A.J., and van der Aalst, W.M.P.: "User-Guided Discovery of Declarative Process Models". In: Proceedings of IEEE Symposium on Computational Intelligence and Data Mining (CIDM), April 2011, Paris, France, pp. 192–199.

[2] Li, J., Bose, R.P.J.C. and van der Aalst, W.M.P.: "Mining Context-Dependent and Interactive Business Process Maps using Execution Patterns". In: Business Process Management Workshops, Lecture Notes in Business Information Processing, Vol. 66, 2011, pp.109-121.

[3] Günther, C. and van der Aalst, W.M.P.: "Fuzzy Mining – Adaptive Process Simplification Based on Multi-Perspective Metrics". In: Business Process Management, LNCS, vol. 4714, 2007, pp. 328-343.

[4] Bose, R.P.J.C. and van der Aalst, W.M.P.: "Abstractions in Process Mining: A Taxonomy of Patterns". In: Business Process Management, LNCS, vol. 5701, 2009, pp. 159-175.

[5] Van der Aalst, W., Reijers,H., Weijters, A., Van Dongen, B., De Medeiros, A., Songa, M., and Verbeek, H.: "Business process mining: An industrial application". In: Information Systems, vol. 32 (5), 2007, pp. 713-732.

[6] Van der Aalst, W., Weijters, T., and Maruster, L.: "Workflow mining: Discovering process models from event logs". In: IEEE Transactions on Knowledge and Data Engineering, vol.16 (9), 2004, pp. 1128-1142.

[7] Van der Aalst, W.M.P., Pesic, M., and Song, M.: "Beyond Process Mining: From the Past to Present and Future". In: Advanced Information Systems Engineering, LNCS, vol. 6051, 2010, pp. 38-52.

[8] Van der Aalst, W.: "Challenges in Business Process Mining". In: Applied Stochastic Models in Business and Industry, 2010

[9] Van der Aalst, W.M.P., Dustdar, S.: "Process Mining Put into Context". In: IEEE Internet Computing, vol. 16, no. 1 (2012) 82-86.

[10] Van der Aalst, W.M.P.: "Process Mining: Discovery, Conformance, and Enhancement of Business Processes", Springer-Verlag, 2011, Berlin-Heidelberg, ISBN 978-3-642-19344-6.

[11] Van Dongen, B.F. and van der Aalst, W.M.P.: "Multi-phase Process Mining: Building Instance Graphs". In: Conceptual Modeling (ER), LNCS, vol. 3288, 2004, pp. 362-376.

[12] Bose, R.P.J.C, Verbeek, E.H.M.W., and van der Aalst, W.M.P.: "Discovering Hierarchical Process Models Using ProM". In: Proceedings of the CAiSE Forum, 2011, London, UK, CEUR Workshop Proceedings, vol. 734, pp. 33-40.

[13] De Medeiros, A.K.A, Pedrinaci, C, van der Aalst, W.M.P., Domingue, J., Song, M., Rozinat, A., Norton, B., and Cabral, L.: "An Outlook on Semantic Business Process Mining and Monitoring".In: On the Move to Meaningful Internet Systems (OTM Workshops), LNCS, vol. 4806, 2007, pp. 1244-1255.

[14] De Medeiros, A.K.A., van der Aalst, W.M.P., and Pedrinaci, C.: "Semantic process mining tools: core building blocks". 16th European Conference on Information Systems, June 2008, Galway, Ireland.

[15] Jablonski, S. and Goetz, M.: "Perspective Oriented Business Process Visualization". In: Business Process Management Workshops, LNCS, vol. 4928, 2008, pp. 144-155.

[16] Volz, B., Zeising, M., and Jablonski, S.: "The Open Meta Modeling Environment". In: Workshop on Flexible Modeling Tools (FlexiTools), May 2011, Waikiki, Honolulu, USA.

[17] Object Management Group (OMG): "Business Process Modeling Notation Version 2.0 – OMG Standard", last access: march 2012.

[18] Weber, B., Sadiq, S., and Reichert, M.: "Beyond rigidity – dynamic process lifecycle support". In: Computer Science - Research and Development, vol. 23, no. 2, 2009, pp. 47-65

[19] Chesani, F., Lamma, E., Mello, P., Montalo, M., Riguzzi, F., and Storari, S.: "Exploiting Inductive Logic Programming Techniques for Declarative Process Mining". In: Transactions on Petri Nets and Other Models of Concurrency II, LNCS, vol. 5460, 2009, pp. 278-295.

[20] Bellodi, E., Riguzzi, F., and Lamma, E.: "Probabilistic Declarative Process Mining". In: Knowledge Science, Engineering and Management, LNCS, vol. 6291, 2010, pp. 292-303.

[21] Horrocks, J., Patel-Schneider, P. F., Boley, H., Tabet, S., Grosof, B., and Dean, M.: "SWRL: A Semantic Web Rule Language Combining OWL and RuleML". W3C Member Submission, last access: March 2012.

[22] Van der Aalst, W. M. P., de Beer, H. T., and van Dongen, B.F.: "Process Mining and Verification of Properties: An Approach Based on Temporal Logic". In: On the Move to Meaningful Internet Systems (OTM Workshops), LNCS, vol. 3760, 2005, pp. 130-147.

[23] Igler, M., Jablonski, S., Günther, C.: "Semantic Process Modeling and Planning". In: SEMAPRO 2010, The 4th International Conference on Advances in Semantic Processing, pp. 199-204, October 2010, Florence, Italy.

[24] Pinter, S., Golani, M.: "Discovering workflow models from activities' lifespans". In: Computers in Industry, vol. 53 (3), 2004, pp. 283-296

[25] Hwang, S., Yang, W.: "On the discovery of process models from their instances". In: Decision Support Systems, vol. 34 (1), 2002, pp. 41-57.

 ISBN: 978-1-61208-223-3