Discovering Data Models from Event Logs

Dorina Bano(✉) and Mathias Weske

Hasso Plattner Institute, University of Potsdam, Potsdam, Germany
{Dorina.Bano,Mathias.Weske}@hpi.de

Abstract. Business process mining is becoming an increasingly important field for understanding the behavioral perspective of any given organization. In a process mining project, process experts are tasked with discovering or improving the operational business processes. They do so by analyzing event logs, the starting point of any process mining endeavor. Despite event logs capturing behavioral information, we argue that they are also a rich source of domain specific information. This information is not represented explicitly in a process model but, nevertheless, it provides valuable contextual information. To this end, we propose a semi-automatic approach to discover a data model that complements traditional process mining techniques with domain specific information. The approach is evaluated in terms of feasibility by being applied to two real-life event logs.

Keywords: Process mining · Event log · Data model

1 Introduction

Process mining is an area of business process management that has taken increased attention from scholars and practitioners in different domains. Some examples of process mining techniques include: process discovery, which aims at discover a process model from the recorded executions of a process; conformance checking, which intends to compare event data with a given process model in order to find deviations; and process improvement, where a business process model is enriched with additional details about its performance [1].

The starting point of any process mining technique is an event log, which is purely a collection of events [2]. In many real-world scenarios such event logs are extracted from data warehouses of a given organizations [3]. After the extraction process takes place the event logs are made available to business process mining experts. Since the log is tailored to discovering and improving a business process the data perspective is usually overlooked. Therefore the process mining experts are left with an event log that does not provide explicit information about the context the data it was extracted from. We argue that the data perspective is an important aspects that complements the process mining procedure with useful information. It plays an important role in the understandability of the event logs and consecutively the process model.

G. Dobbie et al. (Eds.): ER 2020, LNCS 12400, pp. 62–76, 2020.
https://doi.org/10.1007/978-3-030-62522-1_5

In this paper, we introduce a semi-automatic two-step approach for discovering a complementary UML data model from an event log which is tailored for process mining. The discovered data model provides additional insights regarding the domain specific information in the log. In addition, it can be used to enrich the mined process with data objects, therefore, improving its readability.

The reminder of this paper is organized as follows. Section 2 briefly discusses the basic notions needed to understand the rest of the paper. An overview of our two-steps approach is described and illustrated in Sect. 3. Deriving an intermediate representation called activity-attribute relationship diagram (A2A diagram) from an event log is explained in Sect. 4. While the second step of our approach, which discovers a data model from the A2A diagram is depicted in Sect. 5. Section 6 briefly discussed related work. Consecutively, Sect. 7 provides an evaluation of the approach before Sect. 8 concludes the paper.

2 Preliminaries

This section introduces the basic notions and concepts regarding the event log and data model, which we refer to throughout this paper.

2.1 Event Log

Event logs can be extracted from different information systems. Each event log consists of a set of cases. A case is defined as a set of events. Each event involves several attributes such as: the case identifier; the activity name; the timestamp representing the time when the event occurs (i.e., all illustrated in Fig. 1); the resource, i.e. the person or device who executes the activity; the department in which the resource belongs to etc. The first three attributes above are meta-attributes and are mandatory for any event log that is subject to process mining. Let us denote the set of the mandatory meta-attributes with $M_{att} = \{Case, Act, Time\}$. The following definition of event log is based on [1]:

Case	Act	Time	Att1	Att2	Att3	Att4
1	A	24.02.2007	10		1	0
2	A	27.03.2007	7		1	0
2	B	28.05.2007	6	8		0
3	A	01.06.2007	3			0
3	B	19.06.2007	23	2		0

Fig. 1. An example of the event log

Definition 1. *(Event, event log) An event e over a set of attributes Att is defined as $e = (\#_{att_1}, \#_{att_2}, ..., \#_{att_n})$ where $\#_{att_i}$ is the value of attribute $att_i \in Att$ for $i = 1..n$. An event log El is defined as $El = \{e_1, e_2, ..., e_m\}$ where $m \in \mathbb{N}$ is the number of events.*

Definition 2. *(Activity-attribute access relation)*
Let El be an event log and A the set of all unique activities. We define $\#_{att}(e)$, where $att \in Att$, the value of attribute att for event e. We say an activity $a \in A$ accesses an attribute $att \in Att \setminus M_{att}$ iff $\exists e \in El \mid \#_{act}(e) = a \wedge \#_{att}(e) \neq \perp$. Let us denote $r = (a, att, n)$ the access relation between an activity $a \in A$ and an attribute $att \in Att$ in the event log, where $n \in \mathbb{N}$ is the occurrence of the relation in a log. The set of all access relations in the event log is defined as R.

Definition 3. *(No access relation) An activity $a \in A$ does not access an attribute $att \in Att$ iff $\forall e \in El, \#_{act}(e) = a \Rightarrow \#_{att}(e) = \perp$.*

2.2 Data Model

Data model is a fundamental concept for designing and documenting software application. Because of its simplicity, mainly during the design phase, it is used as a mean for communication between the team members. As a target of our approach, the data model is used as a complementary view of the mined business process model to enhance understandability of the use-case scenario.

Below, it is provided a definition of data model withing the scope of this paper:

Definition 4. *A data model is a tuple*
$D = (C, Att, Aso, member, attrmulti, asomulti)$ where:

- *C is a non-empty set of classes*
- *Att is a set of attributes*
- *$Aso \subseteq C \times C$ is a set of associations between classes*
- *$member : C \rightarrow 2^{Att}$ assigns attributes to classes*
- *$attrmulti : Att \rightarrow \mathbb{N}_0 \times \mathbb{N}$ defines the multiplicity of any attribute in a class*
- *$asomulti : Aso \rightarrow \mathbb{N}_0 \times \mathbb{N}$ defines the multiplicity of any association in the data model*

3 Overview of the Data Model Discovering Approach

Before explaining our data model discovery approach let us shortly state the assumptions. We assume that the attribute write access is explicitly represented in the event log, in that for each event it is clear which attributes are written by which activity. For simplicity purpose, we will refer to "write access" as simply "access" for the rest of this paper. In existing event logs, like [4] and [5], the access is represented by concrete values in the accessed attributes and empty

values for the rest of attributes that are not accessed. For example, as it shown in the event log illustrated in Fig. 1, activity *A* access attribute *Att1* three times while activity *B* access the same attribute two times. Meanwhile, *Att2* is accessed only two times by activity *B*. In contrast, *Att2* is only accessed by activity *A* and never by *B*.

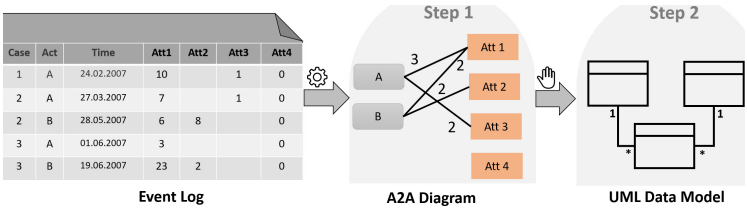

Fig. 2. Overview of the data model discovering approach.

As a pre-processing phase, the event log is cleaned from *null* or *0* values. This implies that we look at activities that in all cases write a certain attribute with null or 0 value. In this case, the activity is considered to not access the given attribute. For example, in the log file shown in Fig. 2 *Att4* is always accessed by both activities with value 0. In this case, we assume that *Att4* is not accessed by any activity as the value 0 may represents an initialization of this attribute or an software error (e.g., default value).

Deriving a data model from the event log implies systematically deriving the individual UML language [6] (our language of choice) constructs: classes, class attributes, and the associations between classes. To this end, we will follow a two-step approach as depicted in Fig. 2. In the first automatic step we introduce a intermediate representation that captures the access relation between all activities and all attributes from the event log. This representation is called Activity-Attribute relationship (A2A) diagram, which is inspired from [7]. In the second step, a set of generic rules are applied to the A2A diagram resulting in the target data model. Afterwards, we leave the choice to the user of the approach to review the generated data model. A detailed explanation of each step is given in the following sections respectively.

4 Derivation of the A2A Diagram

The most prominent information that we have from the event log is the relation of attributes with the activities. To that end, the A2A diagram is derived based on Algorithm 1 as an intermediate step and it is used as input for the data model generation. The starting step for construction such diagram is to identify the activities and attributes from the event log. Therefore, we first derive the set of all activities (at the model level). Second, all attributes, except the meta-attributes (like case identifier, activity name and timestamp), are identified.

The next step for construction of the A2A diagram is to identify the access relation between activities and attributes. Based on Definition 2, if an activity A writes a value to an attribute *Att1*, then there is an access relation between activity A and attribute *Att1* (see Fig. 2). The access occurrence number (depicted over the access arrow in Fig. 2, Step2) represent the number of times an access relation between an activity and attribute holds in the event log independently of the case.

Algorithm 1: A2A diagram derivation from an event log

input : Event log *El*
output: $A2A = (A, Att \setminus M_{att}, R)$ Activity-Attribute access relation diagram
initialization A: empty set of activities, $Att \setminus M_{att}$: set of attributes without the meta-attributes, $R \subseteq A \times Att \setminus M_{att} \times \mathbb{N}$: empty set of access relations `// create a unique set of activities`
[!ht] **for** e *in* *El* **do**
  if $\#_{act}(e) \notin A$ **then**
    add $\#_{act}(e)$ in A ;
  end
end
`// populate the set R`
for a *in* A **do**
  for *att in Att* **do**
    int $n = 0$ `// n represents the access relation occurrence`
    for e *in* *El* **do**
      if $\#_{act}(e) = a \wedge \#_{att}(e) \neq \emptyset$ **then**
        $n = n + 1$
      end
    end
    add $r = (a, att, n)$ in R
  end
end
print(A2A)

5 Data Model Discovery Approach

The data model generation consist in generating the data model classes with their attributes and the associations between the classes. For the data model classes generation we look at the relations between two or more attributes in the A2A diagram and consider whether they belong to the same data model class. After exhaustively going through all the attributes and grouping them into UML classes, we identify the UML associations between those classes. Defining the UML associations entails specifying their multiplicity.

Below we provide a set of rules (see Fig. 3) that are applied to the A2A diagram for grouping the attributes into data model classes. These rules are

organized based on two aspects: not/isolated attributes and not/isolated activities. An attribute is called isolated if all the activity that access it do not access other attributes. Likewise, an activity is isolated if all the attributes it accesses are not accessed by any other activity.

Rule 1: Isolated Attributes, Isolated Activities
We identify the isolated access relations in the A2A diagram, in that an attribute is accessed only by a single activity and the activity accesses only the said attribute. In this case, the rule is to assign all isolated attributes to separate independent UML classes. At this stage, there is no other information in the A2A diagram that can give insights about how the generated UML classes could be related.

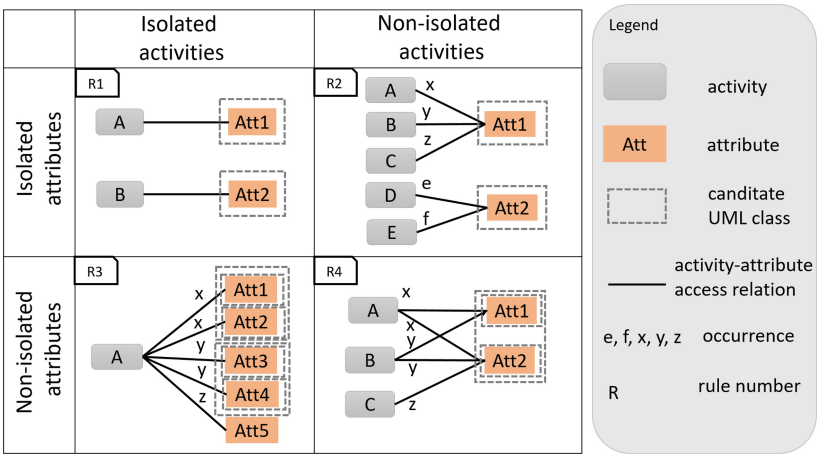

Fig. 3. Rules for deriving data model classes

An example is illustrated in Fig. 3 R1, where activity *A* and attribute *Att1* are isolated because *A* access only *Att1* and *Att1* is accessed by said activity. The same holds for activity *B* and *Att2*. The *Att1* and *Att2* are assigned to a separate UML classes (represented in Fig. 3 by a dashed-line rectangle).

Last, the isolated attributes assigned to the UML class together with their accessing activities are removed from the A2A diagram. This action takes place at the end of each rule.

Rule 2: Isolated Attributes, Non-isolated Activity
In this case, we search the A2A diagram for isolated attributes that are accessed by non-isolated activities. Similar to rule R1, the attributes are isolated and, thus, there is no additional information on how they can be grouped into classes. Hence, the isolated attributes will be each assigned to a separated class.

As it is illustrated in Fig. 3 R2, activity *A*, *B* and *C* are accessing *Att1* with the different cardinalities. The same hold for *Att2*, which is accessed by *D* and *E*. Based on this rule, *Att1* and *Att2* are placed in two independent UML classes.

Rule 3: Non-isolated Attributes, Isolated Activities
If at least one common activity accesses two or more attributes, then the attributes are said to be related. We are looking specifically for related attributes that may belong to the same class. We argue that if an activity accesses two or more attributes with the same occurrence then these attributes are highly likely to be contained in a single class. Therefore, we group these attributes based on common occurrences. However, we cannot deduce from the A2A diagram alone whether the attributes are accessed simultaneously by the activity. It may happen that in total these attributes are accessed the same amount of time by the activity but never in the same event. This means that the attributes are highly likely to not belong to the same class as they seem to be accessed independently. To counter this problem we offer the following solution.

Let E_{A1} be a set of events from the event log where activity A accesses attribute 1. $\mid E_{A1} \mid$ denotes the access occurrence. Similarly, we define E_{A2} as the set of all events where the activity A accesses attribute 2 with occurrence $\mid E_{A2} \mid$, where $\mid E_{A2} \mid \geq \mid E_{A1} \mid$. The decision of whether attribute 1 and 2 belong to the same class is made based on the following function:

$$rel(E_{A1}, E_{A2}) = \begin{cases} \text{one class} & \text{, if } E_{A1} \cap E_{A2} = E_{A1} \\ \text{independent classes} & \text{, if } E_{A1} \cap E_{A2} = \emptyset \\ \text{dependent classes} & \text{, if } 0 < \mid E_{A1} \cap E_{A2} \mid < \mid E_{A1} \mid \end{cases} \quad (1)$$

Attribute 1 and 2 belong to the same class if set E_{A1} is a subset of E_{A2} because anytime the activity A accesses attribute 1 it also accesses attribute 2. Both attributes define a new UML class, however, attribute 1 is marked as optional because it is not always accessed when attribute 2 is accessed.

If the two sets are disjoint (i.e., $E_{A1} \cap E_{A2} = \emptyset$), then attribute 1 and 2 are not in the same class and, moreover, these classes have no association between them. This is due to attribute 1 and 2 happening independently of each other.

Finally, there are events in which attribute 1 and attribute 2 are accessed simultaneously except the first case. This means that there are some events where the attribute 1 and 2 are accessed by the same activity A but this number of events is not the same as $\mid E_{A1} \mid$. In this case, the attributes are placed in different classes, but the classes are still related via an bidirectional association. The multiplicity of the association is 0..1 to * from the class containing attribute 1 to the class containing attribute 2.

In a more general case, where the number of attributes which share the same activity with the same occurrence is more than two, we apply the above function for every pair of attributes to determine the resulting classes.

This rule is illustrated in Fig. 3 R3. Activity A access *Att1* and *Att2* with the occurrence x, *Att3* and *Att4* with the occurrence y and *Att5* with occurrence z. The decision of *Att1* and *Att2* belonging to the same UML class or not depends on whether the events where the activity A access the *Att1* are the same events where the same activity access *Att2*. The same holds for the *Att3* and *Att4* accessed with cardinality y by the same activity.

At last, the attribute assigned to the corresponding classes are removed from the A2A diagram. If their accessing activities do not access other non-isolated attributes, they are removed as well.

Rule 4: Non-isolated Attributes, Non-isolated Activities
Every relation that cannot be expressed by the previous rules is captured by this rule. Activities and attributes are non-isolated, which mean that an attribute is accessed by several activities and each activity accesses several attributes.

After removing the attributes and activities that satisfied the previous three rules we are left with an A2A diagram that contains one or more disconnected subgraphs (i.e., interconnected activities and attributes) which we are referring to as islands. In Fig. 3 R4, there is only one island, but it can happen that another set of non-isolated activities and attributes, which has no relation with the first set, can be left in the A2A diagram. That is why we call these sets islands.

To group the attributes into UML classes each island is decomposed into smaller A2A diagram fragments for each activity. This means that the number of the fragments is the same with the number of activities in an island. The attributes that are accessed by the activity are represented in the respective fragment. Hence, an attribute may appear in one or more fragments (see Fig. 4).

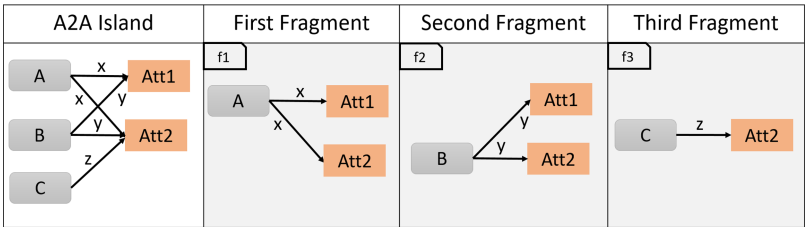

Fig. 4. Decomposed A2A diagram into three fragments after applying the fourth rule

The resulting fragments can satisfy either rule 1 or rule 3 but not rule 2 because the fragments contain only isolated activities. The grouping of the attributes, then, follows the rule 1 or 3. However, since attributes may belong to two or more distinct fragments there is a conflicts that needs to be resolved. For example, it might happen that the same attribute is grouped either in a standalone UML class or in a class with some other attributes depending on the grouping results from each fragment. In this case, we leave the choice to the user of the approach to make a decision that better fits the overall result.

Figure 4 shows an island fragmentation example. Activity A accesses attributes *Att1* and *Att2* with occurrence x. The same holds for activity B except the occurrence, which is y. Last, activity C accesses attribute *Att2* with occurrence z. Based on this rule the A2A diagram is decomposed in three other A2A fragments, one for each activity.

In the first and the second fragment we are dealing with non-isolated attributes and isolated activities. Therefore, rule 3 is applied to derive the classes. In the third fragment, rule 1 is applied because activity *C* and attribute *Att2* are both isolated.

After all the classes are created, the classes can be named based on the activity they were generated from. Finally, we have to consider the associations between the remaining independent classes. To this end, we will consider the most frequently accessed class as the root class, which has the highest potential to represent the business process case notion. Then, we introduce an association between the remaining classes and the root class. Their multiplicities are set based on the occurrences from the A2A diagram.

6 Related Work

In [8] the authors present an approach to obtain a data model from the BPM model, which is then useful for the design phase of the software development process. The authors emphasize that during this phase it is important to use the data model as a common language between the business process analysts and software developers. The focus is on the persistent data rather then the processes' data objects. The authors use a three phase-approach: first, the entities are defined by considering the data stores and the roles played by the participants; second, the relation between entities is deducted based on way the participants and activities manipulate the data store; third, the attributes involved with the participants and data stores are determined. The same direction is followed in the approach presented in [9]. In this paper, we propose an approach that takes as input an event log rather than a business process model. We argue that discovering a business process model from an event log comes with losses in valuable attribute and occurrence information that cannot always be captured by the process model.

Breitmayer et al., [10] propose the discovers of the data model as an intermediate step for discovering object-aware processes. Each table in the database belongs to an object in the data model whereas the database columns represents object attributes. By considering the primary keys and the relation between tables in the database the relations between data objects are defined. The discovered data model is a crucial step for the discovery of the process model. In contrast, our approach relies only in the event log to discover the data model and the data model serves as a complementary artifact to understand the process model.

In [11], the author provides a richer event log, compared to XES, called eXtensible Object-Centric (XOC) by considering multiple case notions called object types. Each event may refer to any number of objects in contrast to the traditional XES format where a single case notion is consider and every event belongs to exactly one case. Constructing a data model from an XES event log is more complicated than deriving it from the XOC format because of the single case notion perspective of XES. XOC holds more information about the data

model because the object relations can be derived from the global perspective (rather than the case perspective) of the events.

There are other approaches, like in [12], that make use of Natural Language Processing (NLP) for deriving a data model from natural language descriptions. The authors argue that such a model is important for the system understandability as it significantly decreases the time needed by a human to understand the system. In this paper, we are introducing a two-step approach by following a set of rules rather than an NLP-based approach, although, the approaches are not mutually exclusive and could be combined to achieve better results.

7 Evaluation

The approach presented in this paper is evaluated based on two real-life event logs, namely: Road Traffic Fine Management (RTFM) [5]; and Sepsis event log [4]. However, for sake of writing space, we describe the evaluation of our approach based on RTFM event log, which is taken from the information systems of the Italian police. The event log contains information regarding the road-traffic fines and includes 150.370 cases (561.470 events) that are processed by the municipality over a three-years time period (January 2010–June 2013). To provide a behavioral overview of the event log, we show in Fig. 5 the process model (represented as BPMN [13]) that is discovered by applying the Inductive Miner algorithm [14]. Some activities that do not access any attribute are excluded from the process model without breaking its meaning.

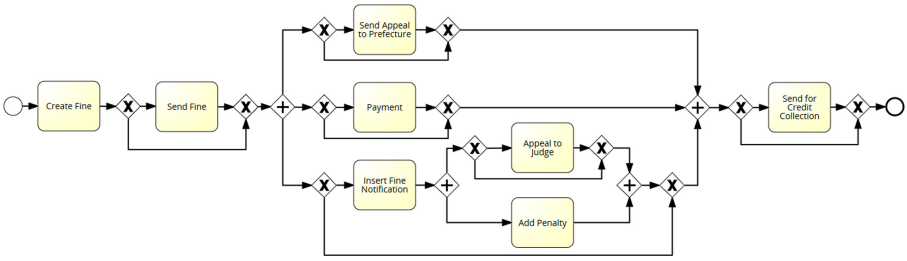

Fig. 5. BPMN model discovered from the RTFM event log

The process starts with *Create Fine* activity. After the fine is created it can be send to the offender via *Send Fine*. The offender has the option to pay the fine immediately after it is handed over to him (*Payment*). If this is not the case, the date when the offender receives the fine is registered (*Insert Fine Notification*). If the payment will not take place (i.e., within 60 days) then a penalty (*Add Penalty*) is added to the fine. The offender has the option to appeal against the fine through the Judge (*Appeal to Judge*) or Prefecture (*Send Appeal to Prefecture*). If the appeal is successful then the process ends. Otherwise the fine

is sent for credit collection (*Send for Credit Collection*) marking the process terminations.

Before generating the A2A diagram the RTFM event log is cleaned from the activity-attribute access relations with the value 0 (i.e., the activity always access the attribute the value 0). For example, *Create Fine* activity access the *Total Payment Amount* always with the value 0. The same holds for *Matricola* and *Resource* attributes accessed by *Appeal to Judge* activity.

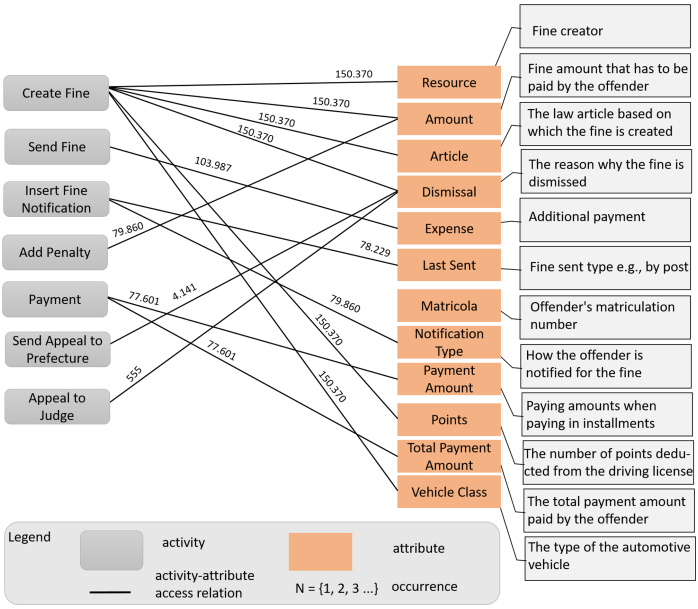

Fig. 6. The RTFM A2A diagram derived by applying Algorithm 1

Applying the first step of our main approach, as that described in Sect. 4, the A2A diagram is generated from the event log (see Fig. 6). The activities that do not access any attribute and all access relations discarded from the clean-up phase are not shown in the A2A diagram. For example, *Matricola* is represented as a stand-alone attribute in Fig. 6 as it is always accessed with value 0.

In the second step, the A2A diagram generated from the event log is used as an input for discovering the data model. Figure 7 depicts the application of the rules from the second step of the approach to the generated A2A diagram. The rules are applied following the defined order (R1 to R4). If a rule is satisfied, all activities and attributes related to that rule are excluded from A2A diagram and the attributes are added in the respective classes. This is repeated until all attributes are grouped into UML classes and there are no attributes left in A2A diagram.

As it is shown in Fig. 7, rule R1 is fulfilled by *Send Fine* activity and *Expense* attribute both represented as isolated in the A2A diagram. Therefore, *Expense*

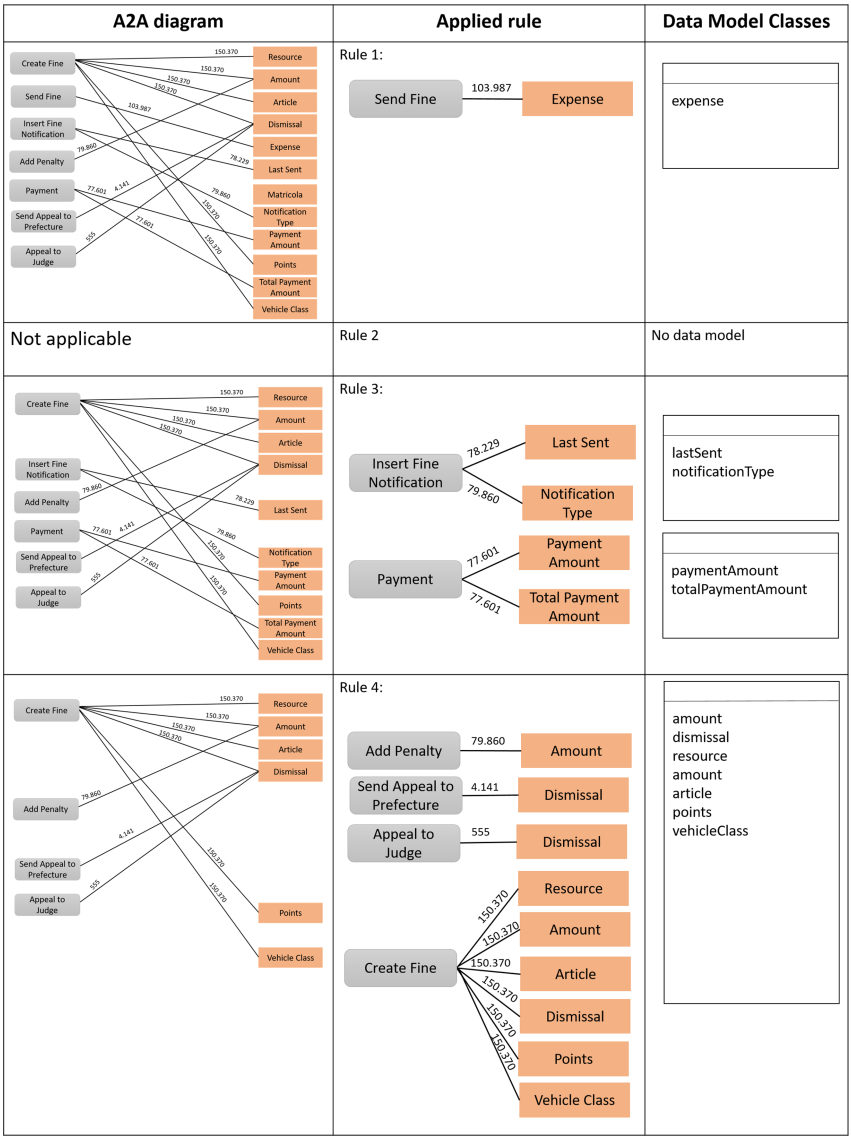

Fig. 7. The approach rules applied to the RTFM A2A diagram

attribute is assigned in separate independent UML classes. Since there is no case of isolated attributes and not-isolated activities in the A2A diagram rule number two does not apply. Subsequently, we check for isolated activities and non-isolated attributes. There are two activities that satisfy the rule R3. First, *Insert Fine Notification* accesses the *Last Sent* and *Notification Type* with different occurrence. In this case, Function 1 is applied to check whether *Insert Fine*

Notification activity is accessing simultaneously both attributes. This happens to be the case in the given log, i.e., in all events where *Insert Fine Notification* accesses the *Last Sent* it also accesses *Notification Type*. Therefore, both attributes are stored in one UML class, where *Last Sent* attribute is marked as optional (based on Function 1). The same holds for *paymentAmount* and *totalPaymentAmount* attributes. Both are simultaneously accessed by the *Payment* activity. Therefore, they are grouped to the same data model class.

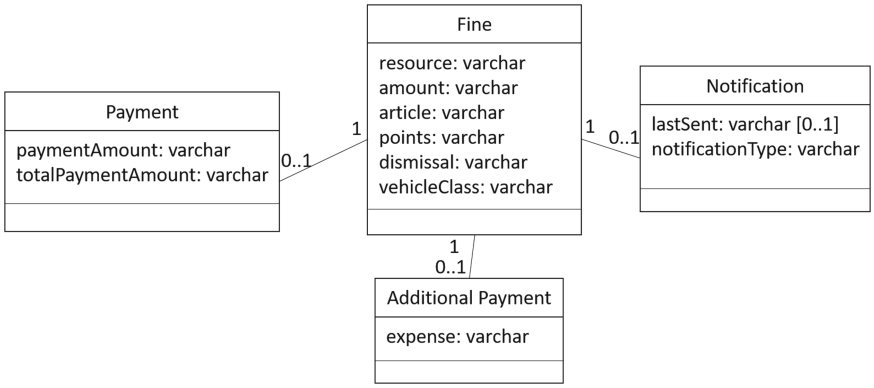

Fig. 8. UML data model generated from the RTFM event log

Lastly, based on rule R4 we check for the non-isolated activities and non-isolated attributes in the derived islands. By applying this rule the A2A diagram is decomposed into four fragments (see Fig. 7, rule 4). In the first three fragments rule R1 can be applied while in the last one rule R3. In this case, the user's choice is to group the attributes in a single class. Fine is the most frequent class therefore is assigned as a root class. After the association between classes are defined the multiplicities are set based on the occurrences from the A2A diagram. The resulting RTFM UML class is depicted in Fig. 8.

8 Conclusion

This paper presents a two-step semi-automatic approach to discover a UML data model from an event log that is purposely designed for process mining. The proposed approach is useful for discovering a data model that complements and increases the understandability of the discoverable process model. The data model contains classes with their attribute, which represent the main entities involved in the process model, and the associations between classes (the relationships between those entities). To achieve this, we consider the relations between activities and attributes in the event log and represent them via an A2A diagram, which is an interim artifact of our approach.

We argue that the discovered data model provides additional insights regarding the domain specific information in the event log. The data model provides complementary information about the entities that are subject to and cannot be captured by a process model.

In future work we plan to extend the approach with NLP solutions to ground the resulting model into a domain specific terminology. In addition, more than one event log from the same organization can be considered to derive a common data model that spans many discoverable business processes.

References

1. van der Aalst, W.M.P.: Process Mining: Data Science in Action, 2nd edn. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49851-4
2. Aalst, W.M.P.: Process mining in the large: a tutorial. In: Zimányi, E. (ed.) eBISS 2013. LNBIP, vol. 172, pp. 33–76. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-05461-2_2
3. Diba, K., Batoulis, K., Weidlich, M., Weske, M.: Extraction, correlation, and abstraction of event data for process mining. Wiley Interdisc. Rev. Data Min. Knowl. Discov. **10**(3) (2020)
4. Mannhardt, F.: Sepsis cases-event log. Eindhoven University of Technology, Eindhoven (2016). https://doi.org/10.4121/uuid:915d2bfb-7e84-49ad-a286-dc35f063a460
5. Mannhardt, F., de Leoni, M.: Road traffic fine management process. Eindhoven University of Technology, Eindhoven (2015). https://doi.org/10.4121/uuid:270fd440-1057-4fb9-89a9-b699b47990f5
6. Rumbaugh, J., Jacobson, I., Booch, G.: Unified Modeling Language Reference Manual, The (2nd Edition). Pearson Higher Education, London (2004)
7. van der Aalst, W.M.P.: Object-centric process mining: dealing with divergence and convergence in event data. EasyChair Preprint no. 2301. EasyChair 2020)
8. Cruz, E.F., Machado, R.J., Santos, M.Y.: From business process modeling to data model: a systematic approach. In: Proceedings of the 8th International Conference on the Quality of Information and Communications Technology, QUATIC 2012, Lisbon, Portugal, 2–6 September 2012, pp. 205–210. IEEE Computer Society (2012)
9. Brdjanin, D., Banjac, D., Banjac, G., Maric, S.: An online business process model-driven generator of the conceptual database model. In: Proceedings of the 8th International Conference on Web Intelligence, Mining and Semantics, WIMS 2018, Novi Sad, Serbia, 25–27 June 2018, pp. 16:1–16:9. ACM (2018)
10. Breitmayer, M., Reichert, M.: Towards the discovery of object-aware processes. In: Manner, J., Haarmann, S., Kolb, S., Kopp, O., (eds.) Proceedings of the 12th ZEUS Workshop on Services and their Composition, Potsdam, Germany, 20–21 February 2020. Volume 2575 of CEUR Workshop Proceedings, pp. 1–4. CEUR-WS.org (2020)
11. Li, G., de Murillas, E.G.L., de Carvalho, R.M., van der Aalst, W.M.P.: Extracting object-centric event logs to support process mining on databases. In: Mendling, J., Mouratidis, H. (eds.) CAiSE 2018. LNBIP, vol. 317, pp. 182–199. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-92901-9_16

12. Meziane, F., Athanasakis, N., Ananiadou, S.: Generating natural language specifications from UML class diagrams. Requir. Eng. **13**(1), 1–18 (2008). https://doi.org/10.1007/s00766-007-0054-0
13. Weske, M.: Business Process Management - Concepts, Languages, Architectures, 3rd edn. Springer, Heidelberg (2019). https://doi.org/10.1007/978-3-662-59432-2
14. Leemans, S.J.J., Fahland, D., van der Aalst, W.M.P.: Process and deviation exploration with inductive visual miner. In: Proceedings of the BPM Demo Sessions 2014 Co-located with the 12th International Conference on Business Process Management, BPM 2014. CEUR-WS.org (2014)