Eventifier: Extracting Process Execution Logs from Operational Databases

Carlos Rodríguez[1], Robert Engel[2], Galena Kostoska[1], Florian Daniel[1], Fabio Casati[1], and Marco Aimar[3]

[1] University of Trento,
Via Sommarive 5, I-38123, Povo (TN), Italy
{crodriguez,kostoska,daniel,casati}@disi.unitn.it
[2] Vienna University of Technology
Institute of Software Technology and Interactive Systems
engel@ec.tuwien.ac.at
[3] Opera21 Group SpA,
Rovereto (TN), Italy
maimar@opera21.it

Abstract. This demo introduces *Eventifier*, a tool that helps in reconstructing an event log from operational databases upon which process instances have been executed. The purpose of reconstructing such event log is that of discovering process models out of it, and, hence, the tool targets researches and practitioners interested in process mining. The aim of this demo is to convey to the participants both the conceptual and practical implications of identifying and extracting process execution events from such databases for reconstructing ready-to-use event logs for process discovery.

1 Introduction

Process discovery is the task of deriving a process model from process execution data that are typically stored in event logs, which in turn are generated by information systems that support the process execution [5]. Most of the approaches available in the state of the art assume the existence of an event log, where each event is assumed to have information, such as a process name, activity name, execution timestamp, event type (e.g., start or end), and process instance ID. In practice, most companies do not really have such an event log, either because they do not have a business process engine that is able to generate such logs or, if they do, the engine supports only parts of the process, e.g., because parts of the process are supported by legacy systems. In the second case, it may also happen that the engine does not generate an event log that can be used for process discovery, e.g., if the log contains only events regarding errors in the system.

The information stored in an event log commonly provides a very narrow and focused view on the overall data produced by a process during its execution (e.g., focusing on errors for recovery or control flow decisions and actors for

auditing). Typically, however, an information system also stores the full data produced by a process inside its ***operational databases*** (OD) (also known as *production databases*), where these data comprise process progression data, process state data, business data produced throughout the process, data related to the regular operations of an organization, as well as their related business facts and objects [2]. ODs therefore store more and richer data than event logs, but blur different aspects of data and neglect the event-based nature of process executions. For this reason, process discovery starts from event logs.

With this demo, we approach the problem of producing process execution events in a fundamentally different context, i.e., in a context where we do *not* have access to the information system running the process (hence we cannot instrument it) and where the only way of obtaining process execution events is deriving them from the OD of the information system *after* the actual process execution. We call this activity ***eventification*** of the OD and we perform it with the help of our tool ***Eventifier***. For the rest of the paper, we assume that the OD is a ***relational database*** [4].

Significance to the BPM field. Much attention has been paid so far to the problems of representing event logs [6], event correlation [3] and process discovery [5], while the problem of how to produce good events has been neglected by research. As explained above, ***Eventifier*** approaches an important issue in the field of process mining by providing an application that will help both researches and practitioners working in the field.

2 Eventification of the Operational Database

Let's start by giving some preliminary definitions. An ***event log*** can be seen as a sequence of events $E = [e_1, e_2, ..., e_m]$, where $e_i = \langle id, tname, pname, piid, ts, pl \rangle$ is an event of a process instance, with *id* being the identifier of the event, *tname* being the name of the task the event is associated with, *pname* being the name of the process type, *piid* being the process instance identifier, *ts* being the timestamp of the event, and *pl* being the payload of the event. Thus, an event log stores traces of process executions as atomic events that represent process progression information and that may carry business data in their payload.

Reconstructing an event log E with events e_i means deciding when to infer the existence of an event from the data in the OD and filling each of the attributes of the event structure with meaningful values. These values either stem from the data in the OD or they may be provided by a domain expert. Specifically, for the *id* attribute, assigning an identifier to an event means recognizing the *existence* of the event. Given that we do not have real events in the OD but other, indirect evidence of their occurrence, there is no "correct" or "original" event identifier to be discovered. The question here is what we consider *evidence* of an event. Similarly, in the case of *tname*, without the concept of task in the applications of the information system, there is no explicit *task naming* that can be discovered from the data. Thus, we need to find a way to label the boxes that

will represent tasks in the discovered model. The value for the attribute *pname* (the *process name*) we can only get from the domain expert, who knows which process she is trying to discover. Then, the process instance identifier (*piid*) is needed to group events into process instances. The *piid* is derived by means of event correlation based on the values of the attributes of the identified events. The attribute *ts* is needed to *order* events chronologically, which is a requirement for process discovery. Therefore, we need to find evidences in the OD that help us in determining the ordering of events. Finally, the goal of choosing a payload *pl* for the purpose of eventification is not to reconstruct the complete *business data* that can be associated with a given task or event, but rather that of supporting the correlation of events into process instances. We can get this data from the rows that originate the events.

We call the assignment of values to *id*, *pname* and *tname* the *identification* of an event, to *ts* the *ordering* of events, to *pl* *data association*, and to *piid* *correlation*. These four activities together constitute the ***eventification*** process, and it is helped by heuristics in the form of ***eventification patterns***:

Event identification patterns. These patterns help in the identification of events from the OD. In these patterns, we assume that the existence of a row in a relation R indicates the presence of an event. We express these patterns as a function:

$$identify(R, pname, tname) \rightarrow e^0 = \langle id, pname, -, tname, -, t \rangle$$

where *pname* and *tname* are defined by the domain expert, and t is the tuple in R that originated e^0. In concrete, we rely on the following three patterns for the identification of events:

- *Single row, single event pattern* (Figure 1(a)). In this pattern, each row in a relation R indicates the existence of an event. R can be obtained with a simple SQL query as:
  SELECT * FROM $r_1, r_2, ..., r_n$
  WHERE [JOIN conditions for $r_1, r_2, ..., r_n$];
- *Single row, multiple event pattern* (Figure 1(b)). A tuple in R can evidence the existence of more than one event, such as when different values of the attributes A_i of R indicate different potential events. In this case, the relation R is built by applying filtering conditions in the WHERE clause so as to keep only the target events:
  SELECT * FROM $r_1, r_2, ..., r_n$
  WHERE [JOIN conditions for $r_1, r_2, ..., r_n$]
  AND [filtering conditions for the target event, e.g., $r_2.dispatched = yes$];
- *Multiple row, single event pattern* (Figure 1(c)). Multiple rows in a relation R indicate the presence of a single event. This last pattern is useful, for instance, when we deal with a *denormalized* relation that mixes data at different granularities, e.g., when in a single tuple we find both the header of an invoice and the item sold. The SQL for R has the following form,
  SELECT DISTINCT $A_1, A_2, ..., A_k$ FROM $r_1, r_2, ..., r_n$
  WHERE [JOIN conditions for $r_1, r_2, ..., r_n$] ;

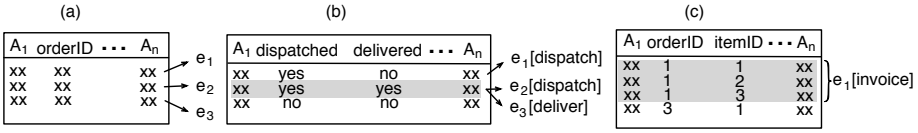

Fig. 1. Types of event identification patterns: (a) single row, single event, (b) single row, multiple events, and (c) multiple row, single event pattern

where the attributes A_i should be the higher granularity attributes that would be typically used in a GROUP BY, SQL statement.

Event ordering pattern. The event ordering pattern aims at deriving the ordering of events from time-related information associated to the records stored in the OD, and is represented as:

$$order(e^0) \rightarrow e^1 = \langle id, pname, -, tname, ts, t \rangle$$

where e^1 is the result of attaching a timestamp value to ts, and ts is the projection of all timestamp or date attributes of $e^0.t$ generated by the previous pattern. If only one timestamp can be found, it is used straightaway. If there are more possible timestamps in pl, the domain expert chooses the one that best represents the execution time of the task.

Data association pattern. The data association pattern aims to select which data to assign to pl. In the above patterns, we have so far simply carried over the complete row t as payload of the event, while here we aim to select which attributes out of the ones in t are really relevant. Our assumption is that all necessary data is already present inside t, that is, we do not need to consult any additional tables of the OD to fill pl with meaningful data. Thus, in the event identification step, the necessary tables are joined, and t contains all potentially relevant data items. The data association pattern is represented as:

$$getdata(e^1) \rightarrow e^2 = \langle id, pname, -, tname, ts, pl \rangle$$

where e^1 is as defined before, and pl is the new payload computed by projecting attributes from t. In absence of any knowledge about the OD by the domain expert, the heuristic we apply is to copy into pl all attributes of t, except timestamps and auto-increment attributes, which by design cannot be used for correlation. The domain expert can of course also choose manually which attributes to include and which to exclude.

Event correlation patterns. Eventually, we are ready to correlate events and to compute the *piid* of the identified events. The goal of event correlation is to group events into process instances, which are the basis for process discovery. As explained above, we assume that after associating the final payloads to events all information we need to correlate events is present in the payload pl of the events in the form of attribute-value pairs. In practice, correlating events into traces means discovering the mathematical function over the attributes of pl that tells

if an event belongs to a given process instance, identified by the output *piid* of the function. We represent this step as follows:

$$correlate(e^2) \rightarrow e = \langle id, pname, piid, tname, ts, pl \rangle$$

where e^2 is as defined above and e is the final version of the discovered event from the OD with the attribute *piid* filled with a suitable identifier of the process instance the event belongs to.

3 The Eventifier Environment

Figure 2 provides an architectural view on the resulting approach to eventification, which is a semi-automated process that requires the collaboration of a domain expert having some basic knowledge of the OD to be eventified. First, the domain expert identifies events in the OD, orders them, and associates data with them. All these activities are supported the the so-called *Event Extractor*, which supports the domain expert in an interactive and iterative fashion. The result of this first step is a set of events, which are however not yet correlated. Correlation is assisted via a dedicated *Event Correlator*, which again helps the domain expert to interactively identify the best attributes and conditions to reconstruct process traces. The result of the whole process is an event log that is ready for process discovery.

The Eventifier is implemented as an integrated platform that includes the components for eventification, correlation and process discovery. These components allow domain experts to interactively apply patterns and to navigate end-to-end from the OD to the discovered process model and back. Since our aim is not to make contributions on process discovery, we use existing process discovery algorithms implemented as plugins for the popular process mining suite ProM [6]. All components are implemented as Java desktop applications using standard libraries such as Swing. The implementation of the Event Correlator is partly based upon a software tool originally developed for the correlation of EDI messages [1]. For the creation of XES-conformant event logs [6] that are used in the interface to process discovery in ProM, we employ the OpenXES libraries (`http://www.xes-standard.org/openxes/start`). Figure 3 shows the screenshots of the Event Extractor and Correlator components.

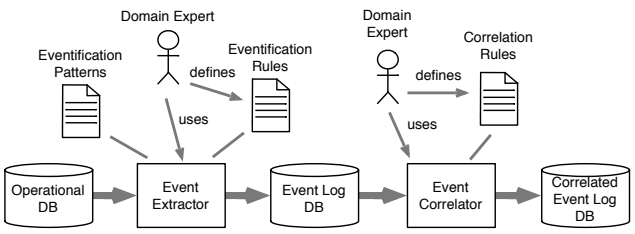

Fig. 2. Overview of the database eventification prototype and approach.

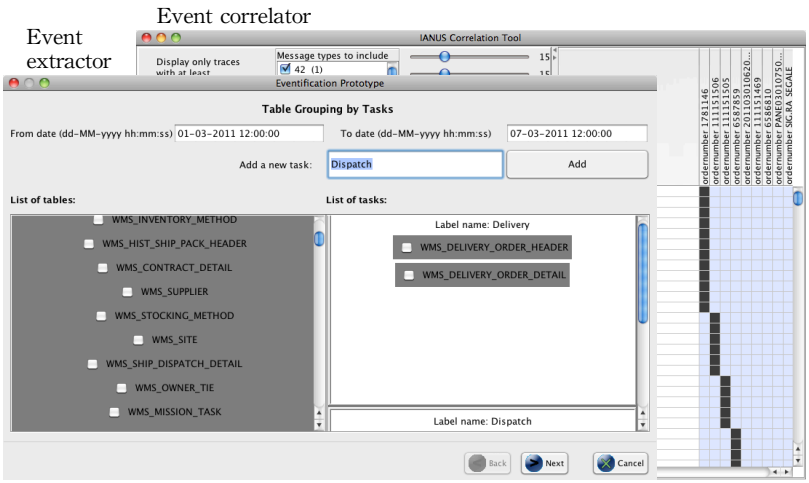

Fig. 3. Screenshots of the components of our integrated platform for eventification.

4 Demo scenario

A demo video of our eventification tool in action can be found at the website `http://sites.google.com/site/dbeventification`. The demo is in the form of a screencast and illustrates the main features of our tool using as scenario the case of an Italian logistics company for refrigerated goods. In this video we clearly show the two main tasks of our approach as outlined in Figure 2 and we also show the final outcome in terms of the process model discovered from the reconstructed event log.

Acknowledgements. This work was supported by the Ianus project funded by the Province of Trento (Italy) and Opera21 Group and by the Vienna Science and Technology Fund (WWTF) through project ICT10-010.

References

1. R. Engel, W. van der Aalst, M. Zapletal, C. Pichler, and H. Werthner. Mining Inter-organizational Business Process Models from EDI Messages: A Case Study from the Automotive Sector. In *24th Int. Conf. on Advanced Information Systems Engineering (CAiSE 2012)*, LNCS 7328, pp.222-237. Springer, 2012.
2. R. Kimball and M. Ross. *The Data Warehouse Toolkit: The Complete Guide to Dimensional Modeling*. Wiley, 2002.
3. H. Montahari-Nezhad, R. Saint-Paul, F. Casati, and B. Benatallah. Event Correlation for Process Discovery from Web Service Interaction Logs. *VLDB Journal*, 20(3):417–444, 2011.
4. R. Ramakrishnan and J. Gehrke. *Database Management Systems*. McGraw-Hill, 2007.
5. W. van der Aalst. *Process Mining: Discovery, Conformance and Enhancement of Business Processes*. Springer, 2011.
6. H. Verbeek, J. Buijs, B. van Dongen, and W. van der Aalst. XES, XESame, and ProM 6. In *Information Systems Evolution*, volume 72, pages 60–75. 2011.