Relational XES: Data Management for Process Mining

B.F. van Dongen and Sh. Shabani

Department of Mathematics and Computer Science,
Eindhoven University of Technology, The Netherlands.
B.F.v.Dongen, S.Shabaninejad@tue.nl

Abstract. Information systems log data during the execution of business processes in so called "event logs". Process mining aims to improve business processes by extracting knowledge from event logs. Currently, the de-facto standard for storing and managing event data, XES, is tailored towards sequential access of this data. Handling more and more data in process mining applications is an important challenge and there is a need for standardized ways of storing and processing event data in the large.
In this paper, we first discuss several solutions to address the "big data" problem in process mining. We present a new framework for dealing with large event logs using a relational data model which is backwards compatible with XES. This framework, called Relational XES, provides buffered, random access to events resulting in a reduction of memory usage and we present experiments with existing process mining applications to show how this framework trades memory for CPU time.

1 Introduction

Over the last few years, the amount of historical execution data stored by large-scale information systems has grown exponentially. This data is typically stored for analysis purposes, for example to enable auditing, process improvement or process mining. The evolution of information system into such large-scale systems increases the need for business process analysis techniques to also handle data at such a large scale.

Most process mining techniques that exist today focus on the analysis of so-called *event logs* with the purpose to discover, monitor and improve business processes. Traditionally, process mining techniques assume that it is possible to *sequentially* record *events* into *traces* such that each event refers to an *activity* (i.e., a well-defined step in the process) and is related to a particular case (i.e., a process instance). Event logs may store additional information such as the *resource* (i.e., person or device) executing or initiating an activity, the *timestamp* of an event, or *data elements* recorded with an event (e.g., the size of an order).

This classic sequential view on a log data has a few important downsides. First, each event is assumed to be relevant for only one trace in the context of one process. In real life this is not necessarily the case, i.e. events may be relevant in the context of many different traces belonging to different processes. Letting an event appear in many traces requires a significant amount of duplication. Second, the tree structure of an event log is great for sequential access to the log, but not suitable for random access, while most

techniques actually use a random access paradigm to access events in the log. Third, the current rise of decomposition and distribution based techniques for process mining requires easy filtering of event logs both vertically, i.e. distributing cases over different logs and horizontally, i.e. distributing events over different logs.

In this paper, we present an architecture that addresses the issues above while it maintains backwards compatibility with existing process mining techniques. Our architecture uses a database to store the event log, allowing for events to appear in multiple traces, for events to be accessed in a random-access fashion and for efficient filtering.

The remainder of this paper is structured as follows. In Section 2 we discuss related work. Then, in Section 3, we present our framework for storing and managing event data. In Section 4 we compare our framework with the de-facto standard for managing event data and we conclude the paper in Section 5.

2 Related Work

Process mining is a research discipline that provides techniques to discover, monitor and improve processes based on event data. It is beyond the scope of this paper to give a full introduction to process mining, but we refer to [7] for such an overview.

Most process mining techniques today have been assuming that event data is available in the form of sequential traces of ordered events that are typically ordered in time. More recently however, process mining researchers have come to realize that events are typically related to multiple processes being executed in parallel while synchronizing on some activities [4–6]. Furthermore, research is emerging on the analysis of streams of events where the notion of a trace is not predefined but may change over time. [1, 3]

Since the start of research on process mining, attempts have been made on standardizing the way in which event data is to be stored. This has lead to a number of semi-standards, such as MXML [9] which was a simple XML format for audit trails of process aware information systems and the recent XES format [10, 11]. The XES format is supported by both academic tools such as ProM [11] as well as industrial tools such as Disco [2]. Many real-life datasets have been made public in the XES xml format

OpenXES is a reference implementation for dealing with event data in the XES format. This reference implementation consists of a number of interfaces as well as a number of concrete implementations. These interfaces allow for both sequential and random, read and write access to event data. Over the years, this implementation has proven to be very successful in the open source process mining framework ProM [11]. Most implementations of OpenXES keep entire logs in memory, while others use internal databases. However, all implementations use XML as their serialization format.

It is not the first time that databases are used to store event logs. XESAme [11] is a log-import framework that allows events from other systems to be converted to a XES file. Internally, this framework uses a temporary database to store event data, but then this database is not exposed to the user and the contents are always serialized to the XES XML format.

3 Relational XES (RXES)

XES is an open standard for *storing* and *managing* event data. For the purpose of storing event data, a standardized, extensible storage format was developed, of which the definition is shown in Figure 1. In XES, each event, trace and log is annotated with typed or untyped attributes which are given semantics through so-called extensions. For example, the activity an event refers to is stored as a literal attribute with key "concept:name". This key is defined in the standard "Concept Extension" [10, 11].

One of the fundamental assumptions of XES is that each event belongs to exactly *one* trace and occurred in the context of exactly *one* log as shown in Figure 1. In practice however, this is often not the case [4].

Therefore, in this paper, we present the RXES framework that lifts this assumption. Instead of considering an event to have occurred in the context of a particular trace, we consider a trace to be a collection of events and a log to be a collection of such traces, but events may occur in many different traces and traces may appear in multiple logs. This allows for backwards compatibility with traditional mining techniques that rely on the "single trace" view, while also allowing for more advanced techniques to consider multiple views at once without the need for duplicating these events.

In Figure 2 we show an ER diagram for the RXES framework. The framework uses a scheme to store events in a database that is largely based on the UML class diagram used for XES but separates the contents of events and traces from their appearance in traces and logs respectively, thus requiring significantly less data duplication. In the remainder of this section, we discuss the main differences between XES and RXES.

3.1 Representation of events and traces

In RXES, logs, traces and events are represented by tables with ids. The actual state of an `XTEvent` is dependent on values of its attributes (accessed through the `XAttributable` class). Similarly, an `XTrace` is nothing more than an ordered list of events (represented by the composition) which carries state through its attributes and can only exist in the context of a log. Therefore, there is no need to have content in the tables representing traces and events in RXES.

Events occurring in the context of a particular trace are represented by the `trace_has_event` table which identifies *occurrences* of events rather than the events themselves and similarly *trace occurrences* are represented by the `log_has_trace` table. As events can appear in multiple traces at different locations and traces can appear in multiple logs at different locations, we keep an order number indicated by `sequence` that is specific to the occurrence of an event in a trace or a trace in a log.

An event appearing in multiple traces can in RXES be represented by a single entry in the event table, but the schema is flexible enough to allow for duplication of the event for each occurrence if desired.

3.2 Representation of Attributes

In XES, attributes are represented by a single class `XAttribute`. Each attribute is composed of a single key and is typed through one of its subclasses (Boolean, Con-

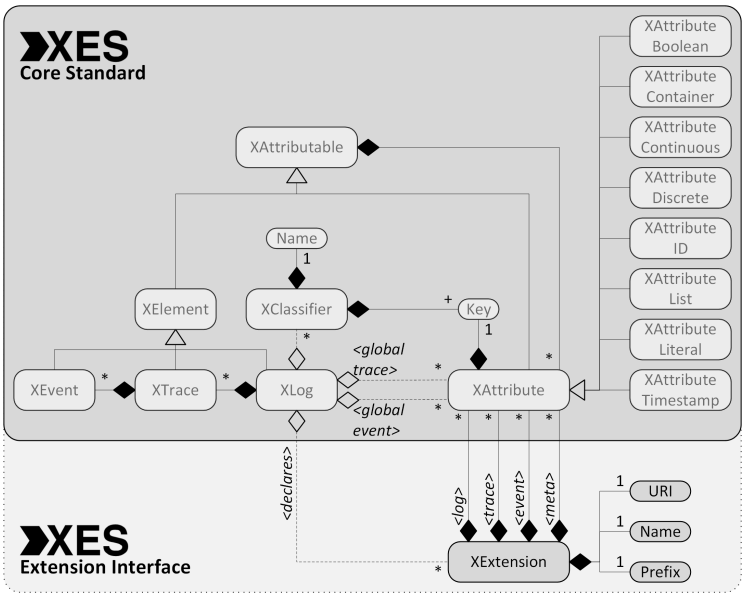

Fig. 1: UML class diagram of the XES standard [10, 11]

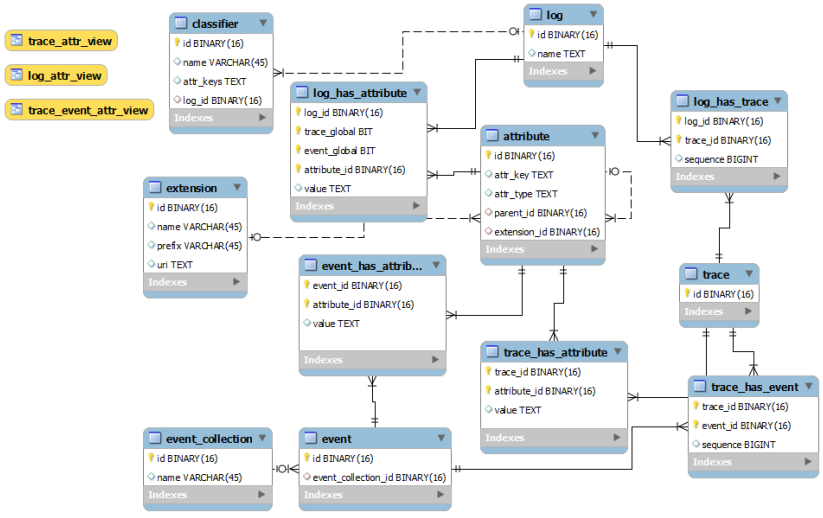

Fig. 2: ER Diagram for RelationalXES

tainer ... Timestamp). The `XAttribute` class may carry attributes itself (i.e. meta-attributes). In RXES, attributes are separated from values to reduce data duplication and to enforce uniqueness of an attribute's type. The latter is technically not a requirement in XES, but it is considered good practice not to use the same attribute with more types. To cover meta attributes the `attribute` table includes a parent child relationship expressing that attributes have attributes. The values of the meta attributes are stored in the `event has attribute` table, hence the definition of a meta attribute that is contained in more than one event attribute with a different value is stored for each value. This is a design choice to avoid having the relate events occurrences with attributes.

As shown in Figure 1, event logs may contain a number of global attributes. The notion of global events refers to the idea that a log can specify that a particular attribute is present on all traces (or events) contained within it, i.e. the attribute is defined to be globally present. This allows for process mining techniques to verify whether a log satisfies certain input conditions, such as the presence of timestamps. A global attribute also specifies a default value which can be used for example when adding new traces or events to a log or in case an attribute declared to be global is nonetheless not present. In RXES, we use two binary attribute in the `log_has_attribute` table to indicate if an attribute is global. To cover the concept of global attribute it has been enforced by the schema that there should not be two global attributes in one log with the same key but a different value.

3.3 Representation of Classifiers and Extensions

Another feature of XES is the availability of so-called classifiers and extensions which provide semantics to events in a log. A classifier in XES is composed of a collection of attribute keys. If a classifier is included in a log, it provides insights into the way individual events should be translated into business activities (or event classes in XES terms). The idea is that such classifiers provide a starting point for process mining techniques to reason on the correct level of abstraction. In practice, classifiers are rarely contained in a log and are often added by the process mining technique. Therefore, in our database, we represent the attribute keys of classifiers simply by text. As a general rule, a classifier should only refer to keys of event global attributes, but this is not enforced by XES nor by RXES.

The last concept of XES that is supported by RXES are the so-called extensions. The extension mechanism allows users to give semantics to attributes. These extensions specify specific keys for attributes which have to be interpreted in a standardized way. For example, the concept extension defined by [10] defines attributes `concept:name` of type `String` which *stores a generally understood name for any type hierarchy element. [...] e.g. the name of the executed activity represented by the event*[10]. In RXES, extensions are stored in a separate table, and attributes are connected to extensions using foreign keys.

3.4 Identification of attributes

The attribute table uses auto-generated IDs attached to each attribute to connect attributes of the same type using SQL queries. By using an id as a primary key rather

than other fields, we allow for recognition of identical attributes which is quite useful for decomposition, filtering and importing. When encountering an attribute that exists in the whole log, we may add a reference to the existing id in the database using an in-memory cache of existing attributes. However, if an attribute is not in cache, we add the attribute using a fresh id. The consolidation of identical attributes can be done offline.

4 Benefits of RelationalXES over OpenXES

RelationalXES provides a full implementation of all OpenXES interfaces using the database as a backend. As a result the framework is fully backward compatible and provides a number of benefits over OpenXES. First, keeping events in the database and only retrieving minimal amounts of data per each request, reduces the memory usage of process mining techniques significantly. Second, through SQL-based filtering and database views, decomposition algorithms no longer require event data to be duplicated in memory, and third, events can exist outside of the context of a trace or a log, allowing RXES to store event data from streams and to utilize trace identification techniques.

RXES has been implemented in Java ,and for the experiments in this paper, MySQL has been used as a database system. However, the design of the application allows for any relational database systems. The framework keeps only the data items that are directly related to the log entity plus list of ids of traces in the memory and later loads the actual data if it is necessary, i.e. if it is requested by a process mining technique. RXES is implemented as a package in the ProM framework and is available as an open-source implementation. It can be downloaded from http://www.processmining.org/.

Currently OpenXES has multiple implementations included in ProM. In this section, we show the difference between our new technique and existing implementations by evaluating memory and CPU usage of processing different sizes of logs.

To investigate memory and CPU usage we used 30 sample event-logs. For that, we used a public, real-life dataset [8] as a base. This dataset contains 13,087 traces with 262,200 events. In total, there are 1,082,719 attributes contained in these events. To investigate the behavior of the system, we extend the size of this log in different dimensions, i.e., we increased the number of events and attributes of the base log up to 10 times . We used a binary search technique to find the least amount of memory needed to process each event log. The resulting memory use for each log and each implementation is shown in Figure 3(a) and Figure 3(b).

Clearly, the default OpenXES implementation (using Java's Collection Framework) has the highest memory allocation. The existing MapDB implementation (in package XESLite) drops the usage by 90%, while our technique uses 90% less than that. For RXES, these results include the memory needed by the Database Management System.

Figures 3(c) and 3(d) show a comparison for the time needed to access all events in the event logs. To evaluate time, a program that requests all elements of the event log has been executed 10 times and an average time is computed. The amount of memory given to the Java virtual machine is sufficient to meet the demands of the Default OpenXES implementation. Clearly, there is a direct relation between memory usage and speed of access. As Figure 3(c) shows, the default implementation is the fastest method simply because the event log is loaded into memory completely. The figures further show that

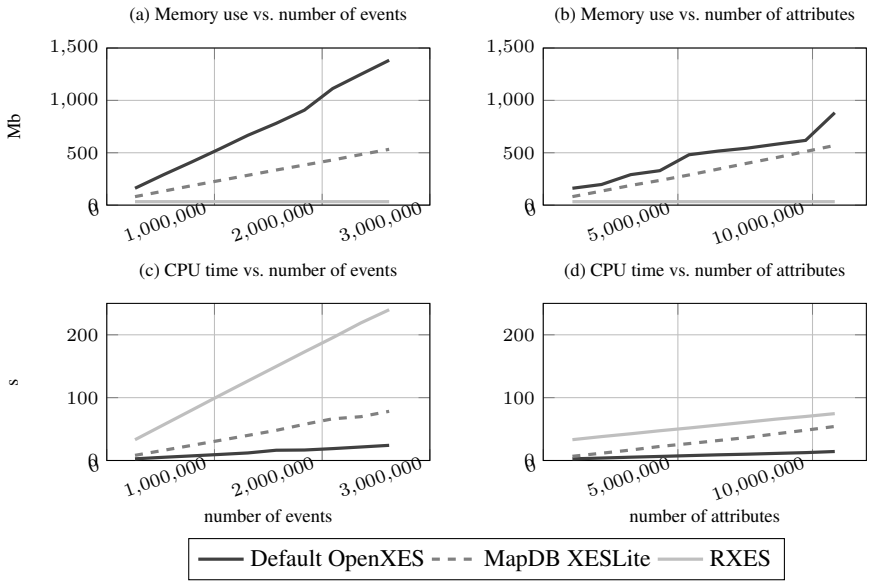

Fig. 3: Memory and CPU use for accessing all events in a log vs. number of events and attributed in the log.

RXES is slower than MapDB which uses an in-memory database, which is mostly because for this experiment we used minimum size of the trace buffer which only keeps one trace at a time. Hence, the execution time is result of performing many SQL queries over a TCP/IP connection to the DBMS.

In general, if the size of the event log is small compared to the available memory, MapDBDisk keeps a good balance between memory and time usage. However, compared to RXES, MapDB doesn't allow for distribution and replication, while it is vital in cases of using decomposition and parallelism to handle the big data.

The ability use of RXES to process large event logs with readily available process mining techniques is essential for the adoption of our framework in practice. However, our framework is capable of performing tasks directly on the database, such as event log filtering.

5 Conclusion and Future Work

In this paper, we presented RelationalXES. This framework is a generalization of the de-facto standard XES for storing and managing event data in process mining. Where all existing implementations of XES uses the strict notion of containment for events in traces and traces in logs, our framework is much more flexible and allows for events to appear in more than one trace and for traces to appear in more than one log. That makes it possible to have different views on the same event log. Furthermore, the database

schema used in RXES allows for a significant reduction of duplication by storing frequently occurring attributes only once rather than repeating them for every occurrence.

In the paper, we presented the framework in detail and we discuss the underlying database schema. Furthermore, we show the reduction of memory use by process mining techniques using RXES.

References

[1] Andrea Burattin, Alessandro Sperduti, and Wil M. P. van der Aalst. Heuristics miners for streaming event data. *CoRR*, abs/1212.6383, 2012.

[2] Christian W. Günther and Anne Rozinat. Disco: Discover your processes. In Niels Lohmann and Simon Moser, editors, *BPM (Demos)*, volume 940 of *CEUR Workshop Proceedings*, pages 40–44. CEUR-WS.org, 2012.

[3] Fabrizio Maria Maggi, Andrea Burattin, Marta Cimitile, and Alessandro Sperduti. Online process discovery to detect concept drifts in ltl-based declarative process models. In Robert Meersman et.al.s, editor, *OTM Conferences*, volume 8185 of *Lecture Notes in Computer Science*, pages 94–111. Springer, 2013. ISBN 978-3-642-41029-1.

[4] Erik H. J. Nooijen, Boudewijn F. van Dongen, and Dirk Fahland. Automatic discovery of data-centric and artifact-centric processes. In Marcello La Rosa and Pnina Soffer, editors, *Business Process Management Workshops*, volume 132 of *Lecture Notes in Business Information Processing*, pages 316–327. Springer, 2012. ISBN 978-3-642-36284-2.

[5] Viara Popova and Marlon Dumas. Discovering unbounded synchronization conditions in artifact-centric process models. In Niels Lohmann, Minseok Song, and Petia Wohed, editors, *Business Process Management Workshops*, volume 171 of *Lecture Notes in Business Information Processing*, pages 28–40. Springer, 2013. ISBN 978-3-319-06256-3.

[6] Viara Popova, Dirk Fahland, and Marlon Dumas. Artifact lifecycle discovery. *CoRR*, abs/1303.2554, 2013.

[7] Wil M. P. van der Aalst. *Process Mining - Discovery, Conformance and Enhancement of Business Processes*. Springer, 2011. ISBN 978-3-642-19344-6.

[8] B.F.; van Dongen. Bpi challenge 2012, 2012. URL `http://dx.doi.org/10.4121/uuid:3926db30-f712-4394-aebc-75976070e91f`.

[9] Boudewijn F. van Dongen and Wil M. P. van der Aalst. Emit: A process mining tool. In Jordi Cortadella and Wolfgang Reisig, editors, *ICATPN*, volume 3099 of *Lecture Notes in Computer Science*, pages 454–463. Springer, 2004. ISBN 3-540-22236-7.

[10] H. M. W. Verbeek and Christian W. Günther. XES standard definition 2.0. Technical report, BPMcenter.org, July 2014. URL `http://bpmcenter.org/wp-content/uploads/reports/2014/BPM-14-09.pdf`. BPM Center Report BPM-14-09.

[11] H. M. W. Verbeek, Joos C. A. M. Buijs, Boudewijn F. van Dongen, and Wil M. P. van der Aalst. XES, XESame, and ProM 6. In Pnina Soffer and Erik Proper, editors, *CAiSE Forum*, volume 72 of *Lecture Notes in Business Information Processing*, pages 60–75. Springer, 2010. ISBN 978-3-642-17721-7.