Article

Towards a Domain-Specific Modeling Language for Extracting Event Logs from ERP Systems

Ana Pajić Simović *, Slađan Babarogić, Ognjen Pantelić and Stefan Krstović

Faculty of Organizational Sciences, University of Belgrade, 11010 Belgrade, Serbia;
sladjan.babarogic@fon.bg.ac.rs (S.B.); ognjen.pantelic@fon.bg.ac.rs (O.P.); stefan.krstovic@fon.bg.ac.rs (S.K.)
* Correspondence: ana.pajic@fon.bg.ac.rs

Abstract: Enterprise resource planning (ERP) systems are often seen as viable sources of data for process mining analysis. To perform most of the existing process mining techniques, it is necessary to obtain a valid event log that is fully compliant with the eXtensible Event Stream (XES) standard. In ERP systems, such event logs are not available as the concept of business activity is missing. Extracting event data from an ERP database is not a trivial task and requires in-depth knowledge of the business processes and underlying data structure. Therefore, domain experts require proper techniques and tools for extracting event data from ERP databases. In this paper, we present the full specification of a domain-specific modeling language for facilitating the extraction of appropriate event data from transactional databases by domain experts. The modeling language has been developed to support complex ambiguous cases when using ERP systems. We demonstrate its applicability using a case study with real data and show that the language includes constructs that enable a domain expert to easily model data of interest in the log extraction step. The language provides sufficient information to extract and transform data from transactional ERP databases to the XES format.

Keywords: event log; transaction log; ERP system; artifact-centric approach; business document

check for updates

Citation: Pajić Simović, A.; Babarogić, S.; Pantelić, O.; Krstović, S. Towards a Domain-Specific Modeling Language for Extracting Event Logs from ERP Systems. *Appl. Sci.* **2021**, *11*, 5476. https://doi.org/10.3390/app11125476

Academic Editors: Jaehun Park and Hyerim Bae

Received: 30 April 2021
Accepted: 7 June 2021
Published: 12 June 2021

1. Introduction

At present, organizations use enterprise resource planning (ERP) systems as a core information and communications technologies (ICT) component. ERP systems encompass configurable modules supporting business processes that have received wide industrial adoption, and they contain extensive amounts of data about the behaviors of such processes. Recorded data can be used to analyze whether a predefined process specification conforms with real activities [1]. Process mining offers automated techniques for this type of analysis. Process mining has emerged in the past few years as a new analytical discipline that focuses on extracting insights about processes from the event data stored in information systems.

Process mining provides the techniques to automatically discover process models from data, find the mismatch between process models and process executions, and improve models based on information obtained from their past executions [2]. These techniques rely on the existence of an event log whose structure is suitable for mining. An event log assumes that events that refer to an activity occur at a particular time in precisely one case [3]. While event logs structured in this form are mostly available in process-aware information systems, such a structure is not explicitly generated by ERP systems and other domain-specific business systems [4–6]. These systems record events implicitly, separately, and without a common case identifier. Thus, such logs need to be generated from the data that typically exist in a relational database. To build such an event log is the key enabler for process mining and is a complex task [7,8].

To extract data from ERP systems in an event log format suitable for process mining, several challenges need to be addressed. First, there is not one possible case notion, but multiple case notions (called object types) which could be used to correlate the corresponding events in traces [9]. For example, a purchase order or purchase invoice document can be identified as a case notion for a purchasing process in an ERP system. Second, data are organized in documents related to each other through one-to-many and many-to-many relationships, and activities are often related to these documents. Due to the data featuring one-to-many and many-to-many relations, an activity can be performed multiple times on a single document or performed only once but on different documents; however, existing process mining techniques and tools require an XES event log as an input. The eXtensible Event Stream (XES) format [10] is the current Institute of Electrical and Electronics Engineers (IEEE) standard for storing events and requires a single case notion to correlate events and precisely one case per event. Thus, the resulting log may suffer from convergence (one event is related to multiple cases) and divergence (multiple events of the same type are related to one case) when extracting an event log from the ERP system, as described in [11]. This may lead to duplicated events and false data dependencies between events.

This paper emphasizes the challenge of building an event log from ERP systems supporting domain knowledge. Extracting events from an ERP database requires substantial domain knowledge of the business processes and underlying data model [2]. Data constraints in ERP systems are generally managed at the application level [12,13]. In this situation, when data relations are defined outside a database schema, domain knowledge is essential to select the correct data from database tables. The semantics of event data and where they are stored within an ERP system is known by domain experts. Jans and Soffer [14] specified a set of decisions that domain experts often need to make when creating event logs from databases. They have discussed how these decisions can affect log quality. One of these decisions is the selection of a meaningful case notion. This shows that the need to support domain experts during the log building process is a real issue, but there is an absence of techniques and tools to facilitate this [9,15].

To support domain experts, to automate log building process as much as possible, and to mitigate the problems caused by many-to-many relations, a graphical domain-specific modeling language (DSML) for extracting event data from transactional databases has been developed. It has been specifically designed to describe behaviors for complex data in ERP systems in terms of multiple interacting artifacts. To this end, we have followed artifact-centric approaches [11,16] which consider artifacts, i.e., the documents of ERP systems, as case identifiers. These approaches can deal with one-to-many and many-to-many relations between data.

A few researchers have also attempted to address the problems caused by many-to-many relations and have proposed novel metamodels for storing data for process mining [13,17] and novel modeling notations and process mining techniques for discovering and analyzing artifact-centric processes [18,19] to achieve this; however, our proposed metamodel is aimed at providing the basis for a data extraction model to create event logs in the XES format from transaction logs. Once event logs are obtained, i.e., XES files, it is possible to perform existing process mining techniques which users are already familiar with. The language supports domain experts to determine the process scope, select correct database tables, and include relevant events for conversion definition in an intuitive way.

The metamodel proposed in this paper, along with its corresponding constraints, is the result of several iterations of development and revision. We also introduce a graphical notation for the language concepts. Once the DSML model was designed, the next step of our approach was to transform the model into Structured Query Language (SQL) code. Then, automatic translation from the query results to the final XES event log could be performed. The main contribution of our research is the full specification of the aforementioned language. The benefit of designing a specific language is that it allows domain experts, even without any programming background, to focus on modeling the data of interest in the log extraction step. Another important contribution is a conceptual representation of ERP database content in terms of business artifacts. The new metamodel aligns behavioral and data perspectives in order to provide sufficient information about a process, data objects, and the relations between them. Based on this model information and the specified transformation rules, it will be possible to extract and transform data from transactional ERP databases to the XES target representation, as demonstrated in this paper.

The remainder of the paper is organized as follows. The next section discusses related works. Section 3 describes complex ambiguous cases from ERP systems. The specification of the language, its metamodel, and the corresponding constraints are discussed in Section 4. Besides, a graphical notation is presented. Then, our solution is evaluated using a case study with real data in Section 5. A discussion is presented in Sections 6 and 7 provides final conclusions and ideas for future work.

2. Related Work

Various techniques and tools in the literature have tried to deal with the extraction of event logs from enterprise systems. Tools such as XESame [20] and ProMimport [21] can be used to extract XES event logs from various data sources that hold explicit information about process instances and events. The XES log format has been accepted by the IEEE Task Force on Process Mining as a de facto standard for process mining algorithms. This format represents a structure of an event log, containing traces and events with the corresponding attributes. It comes with standardized extensions for capturing semantics to particular attributes per component. Briefly, a trace describes an order in which events occur during one execution of a process. Note that traces in the XES log describe the evolution of the main case object, i.e., one type of process instance. Moreover, each event refers to some activity. We refer to the official IEEE XES standard [10] for more information about XES metamodel and the OpenXES reference implementation.

Since ERP systems provide several identifiers for a process instance, the XES log format cannot be extracted directly. So far, most studies in this regard have dealt with extracting data from specific environments like SAP systems and have not provided a general approach for building event logs from databases. Ingvaldsen and Gulla [22] developed a tool to specifically construct process chains from SAP transactions while allowing users to define events. Piessens [23], in turn, initialized a repository with predefined event types for specific processes supported by SAP. A more generic solution is the onprom project presented by Calvanese et al. [5], who introduced two ontologies, namely, a domain ontology and an event ontology, to support the extraction of event logs from legacy data. Specifically, the approach provides an annotation mechanism to help users in the conceptual identification of event data. This approach addresses the data from a semantic perspective but does not deal with one-to-many and many-to-many relations, as encountered in this work. Another effort to obtain event logs considered the redo log files of relational databases as a source of event data [4,24]. Redo logs are used to record information about any modification performed on the data of the database; however, more files need to be explicitly configured to store redo logs. In many real cases, redo logs might not be available.

The OpenSlex metamodel [13] has been proposed to store object-centric event logs, covering different types of entities such as objects, object versions, relations, and events. Multiple XES event logs can then be created from this intermediary storage based on the chosen case notion. Unlike OpenSlex, our solution provides the basis for a data extraction model and does not require the manual specification of database queries. It abstracts users from the definition of complex transformations required to obtain an XES event log. Once a conceptual model has been designed by domain experts, it provides sufficient information to automate the generation of queries and translate the obtained query results to the final XES event log.

A more recent work [19] has proposed a new process mining technique to discover an object-centric behavioral constraint (OCBC) model from an eXtensible object-centric (XOC) log. This technique is based on a novel log format and a new type of model which is used to describe processes involving complex data dependencies. The XOC format does not assume a unique case notion to group events, thus, it can be extracted from object-centric data [17]. The new log format combines different types of entities, such as objects, events, and attributes, as presented in [13]. It provides additional information, such as database objects, in order to solve data convergence and data divergence problems; however, the major drawback of an object-centric approach is that it forces the representation of complex systems by means of a novel log format, which is not widely accepted in the industry when compared with the IEEE XES standard. This necessitates new process mining tools and existing process mining techniques cannot be used. Domain experts cannot easily interpret OCBC models discovered from XOC logs as they have constructs that are different than conventional process modeling notations such as the BPMN or Petri net notations [15]. Our aim is to enable the building of an event log that could be managed by the existing tools used in process mining.

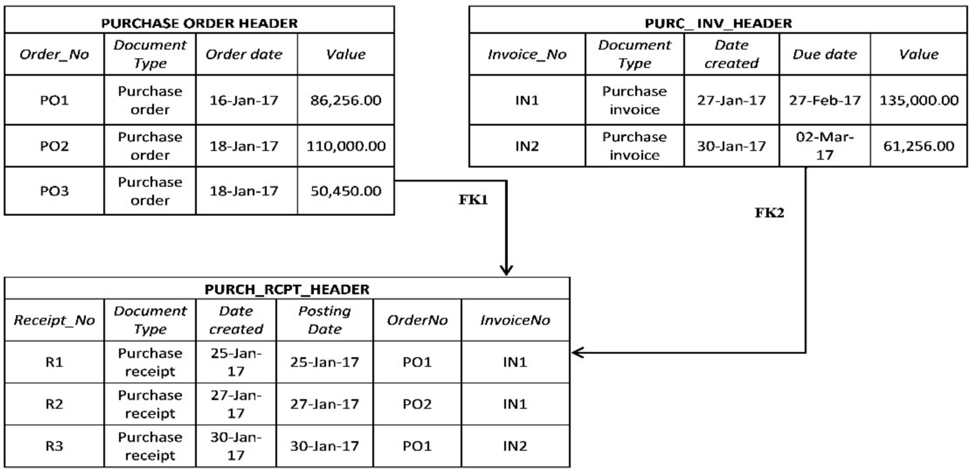
Our work also relates to artifact-centric approaches [11,16,25]. These approaches identify data artifacts, i.e., documents like a purchase order, in order to discover the life cycles of objects with one-to-many and many-to-many relations between them. For each data artifact, its instances and all related events are extracted and transformed into a separate event log. As a result, multiple event logs are obtained. The benefit of applying an artifact-centric approach is that it addresses the data convergence and data divergence problems explicitly by defining an instance notion in each artifact; however, more support should be provided for domain experts to identify the essential artifacts for data extraction from ERP systems. Our approach attempts to address this issue. Unlike existing approaches, we propose a metamodel for describing the semantics of transactional data from an ERP system. Inspired by artifact-centric approaches, the data structure of the process is conceptually represented by multiple related documents and their transactions. As far as we know, this is the first work that uses domain-specific language for obtaining event logs from object-centric information systems such as ERP systems. The benefit of designing a specific language to support data extraction is that it allows the domain experts to localize the correct data for the process mining analysis in different tables and data structures of ERP system at the higher level of abstraction, thus avoiding implementation details. This consequently guides a domain expert towards converting data from a database to a suitable event log without the need for programming.

3. Motivation and Running Example

An ERP system is a business suite of integrated applications that supports the major business processes of an organization. Typical examples of such processes in ERP systems are purchase order fulfillment and manufacturing. The key characteristic of an ERP system is that the different types of documents are used to record transaction data that are often tied to the execution of process activities. For instance, material documents record goods that are either received into or issued from inventory. Document types are implemented as table pairs with parent-child relationships between tables. This can be represented as a header and lines pattern, where the header table holds general information of a document and the lines table captures specific line item information of a document. The header and lines design pattern is very common for business applications that deal with documents, such as ERP, customer relationship management (CRM), and supply chain management (SCM) systems.

In ERP systems, the process logic and business rules are both embedded in the application code managing documents in typically many-to-many relationships [13,26]. The presence of these complex data dependencies has a large influence on the definition of cases for process mining analysis. When a process has one-to-many and many-to-many relations between documents, the user has to choose one document as a case notion as multiple documents can be linked to a single process execution. Experience teaches us that the ability to obtain the event log, which accurately represents reality, depends on this decision.

Here, we introduce an ordering process as a running example to illustrate the complex data dependencies in an ERP system. In general, the process involves supplying goods or services, receiving supplies, and processing invoices. The process starts with creating a purchase order which is forwarded to the selected vendor. When the vendor delivers supplies, a purchase receipt document is created and an invoice for the payment is registered. Figure 1 gives a simplified example data set from the ordering process supported by the Microsoft Dynamics NAV ERP system. This example of transactional data describes the event type of creating a document. Note that multiple invoices can be received for a single order (IN1 and IN2 refer to PO1) and one invoice can be matched to multiple purchase orders (IN1 refers to PO1 and PO2).



PURCHASE ORDER HEADER			
Order_No	*Document Type*	*Order date*	*Value*
PO1	Purchase order	16-Jan-17	86,256.00
PO2	Purchase order	18-Jan-17	110,000.00
PO3	Purchase order	18-Jan-17	50,450.00

PURC_ INV_HEADER				
Invoice_No	*Document Type*	*Date created*	*Due date*	*Value*
IN1	Purchase invoice	27-Jan-17	27-Feb-17	135,000.00
IN2	Purchase invoice	30-Jan-17	02-Mar-17	61,256.00

PURCH_RCPT_HEADER					
Receipt_No	*Document Type*	*Date created*	*Posting Date*	*OrderNo*	*InvoiceNo*
R1	Purchase receipt	25-Jan-17	25-Jan-17	PO1	IN1
R2	Purchase receipt	27-Jan-17	27-Jan-17	PO2	IN1
R3	Purchase receipt	30-Jan-17	30-Jan-17	PO1	IN2

Figure 1. A simplified example of a purchasing process.

As already mentioned, a process instance represents the unique document or case object that is followed throughout the process. An invoice, a receipt, or purchase order can be selected as a process instance, i.e., case notion. If a purchase order is selected as the process instance, the trace of process instance PO1 reveals the link from the book invoice to the posting goods receipt activity in the discovered process model shown in Figure 2. There are two posting goods receipt events (R1 and R3) and two book invoice events (IN1 and IN2) for the process instance (PO1). Due to the multiple instances of the same activity referring to the single process instance, the link may indicate the false dependency between the events. For instance, we can infer that there are invoices registered before the related purchase order is received; however, receipt document R3 is created before invoice document IN2, but this is not visible in the discovered process model. This situation corresponds to the data divergence problem.

The data convergence problem is a result of a many-to-one relationship between the document that represents a process instance and its related document, and the problem is related to the number of times an activity is performed. For instance, invoice IN1 is matched to two different purchase orders (PO1 and PO2). Within the ERP system, invoice IN1 is registered only once but is extracted twice and the event is duplicated in the process model discovered using traditional process mining techniques. For the same activity, one event per process instance is observed in the discovered process model in Figure 3. The number in brackets represents how many times the activity is performed.

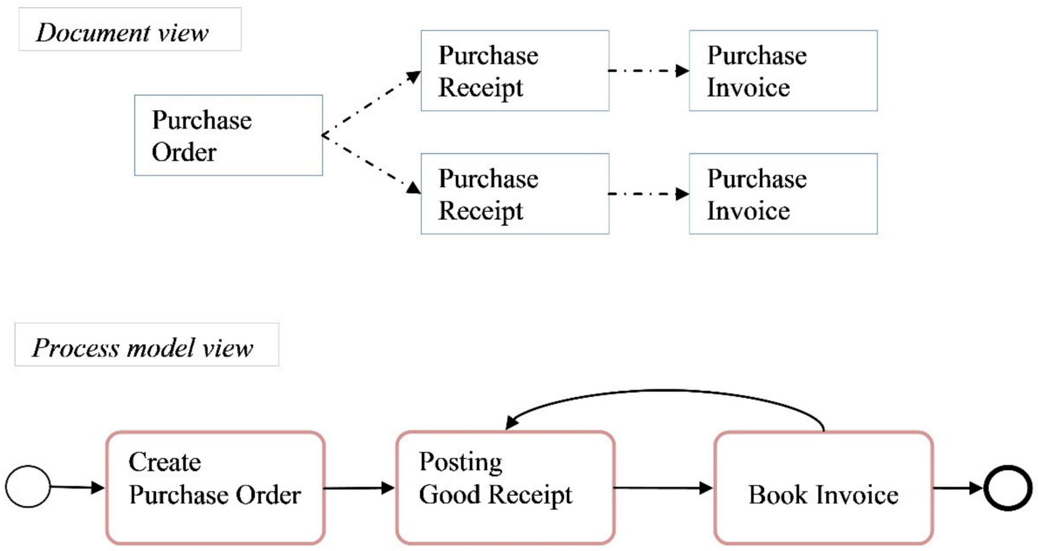


Figure 2. Data divergence problem as visualized from different views.

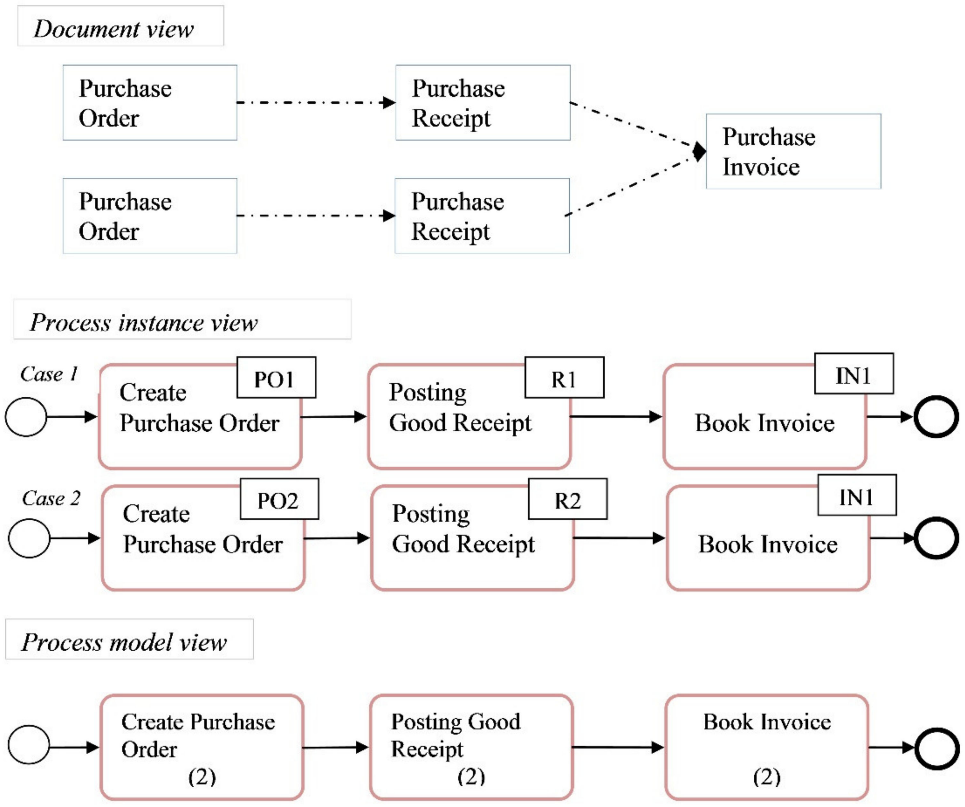


Figure 3. Data convergence problem as visualized from different views.

4. Domain-Specific Modeling Language for Extracting Event Logs

This section presents the definition of the abstract syntax as a metamodel, and a graphical concrete syntax is described. The language can be used in a pre-processing step for process mining projects to extract correct data from ERP systems for the analysis in a user-friendly way. It is aimed at structuring and integrating the elements of the process and data perspectives in a single structure in order to support an artifact-centric approach.

4.1. Abstract Syntax

The specification of the essential domain concepts and relationships between them is the central part of designing a DSML. It is composed of a metamodel, where the attributes and associations of the language concepts are defined, as well as the corresponding constraints. We chose to specify restrictions and semantic properties of the DSML elements more through the abstract syntax than by adding a larger number of model constraints. By defining fewer additional constraints, the comprehensibility and readability of a metamodel can be improved. Also, the correctness of a model can be checked at the class level [27].

To define the abstract syntax of a language, a metamodel has been created using the Unified Modeling Language (UML) class diagram [28]. A UML class diagram used as its metamodeling language is easier to understand by domain experts and the format is widely accepted for conceptual modeling in industrial fields. The UML metamodel is compliant with the Meta-Object Facility (MOF) standard [29]. Figure 4 shows the main metaclasses in our metamodel, along with their attributes and relationships. The metamodel refers to the existing structure of the XES standard. The XES standard enforces the general structure of an event log containing the log, trace, event, and attribute objects. All information is stored in attributes where the semantics are clearly defined using extensions. It is possible to enrich the format with new domain-specific attributes for standard entities; however, extensions cannot be used to include new entities and structures to store additional information. The data perspective is missing in the XES, thus, the new metamodel proposed in this work describes necessary elements related to data objects within a process flow that are interesting for process mining analysis.

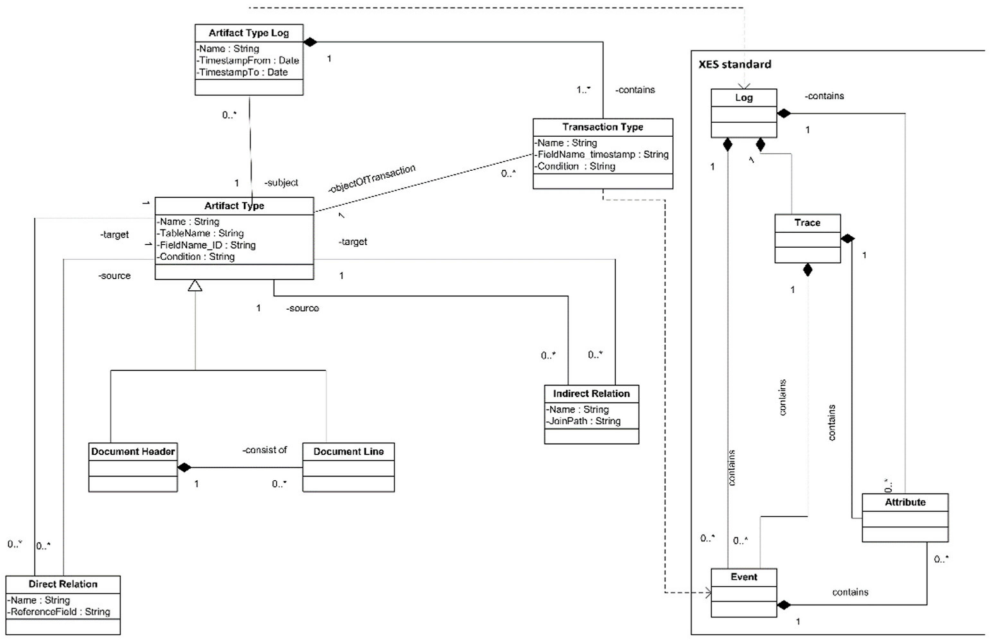


Figure 4. Abstract syntax of the DSML for supporting event log extraction from ERP systems.

The main construct of the metamodel is the artifact, which refers to the key business objects, i.e., documents. A business document structure typically contains a header and a set of lines, such as a purchase order header and line item detail. A parent-child relationship is used to associate one document header with its many lines in a database. A document header table contains general information of a transaction, while a document line captures specific line item properties. Note that transactions refer to operations on a complete document or only to a line of a document. A purchasing transaction, for example, to approve an order is linked to a purchase order header. Hence, both the header and line are subsumed in the abstract artifact entity type, which can be associated with several transactions by means of the relationship called the "object of transaction" in the metamodel. Each transaction in turn belongs to exactly one artifact.

The artifacts are connected to each other via direct or indirect relations. A relation holds a link from a source artifact to a target artifact. A direct relation specifies the reference linking the two artifacts, while an indirect relation is constituted of two or more direct links. This is performed using an attribute called "join path".

We have only represented the main metaclasses of XES metamodel to facilitate the comprehension of relations between the language elements and the elements of XES. The elements are linked using dependency relationships. This relationship indicates the dependencies between an XES element and the element of the presented metamodel for transformation implementation. The XES log element will use the artifact type log metaclass for defining a case notion to correlate events. The artifact type log metaclass has a "main artifact ID" property referencing the artifact element whose instances can be related to XES trace elements. There may only be one main artifact, and it represents a case notion. Each event in a log can be related to only one occurrence of a transaction, which is indicated by the dependency relationship between transaction type and XES event element.

The constraints that impose certain restrictions on how to construct a model are described in the following:

- Artifact type log: The artifact type log is the main metaclass that represents the root of every model. The main artifact of the log must be specified, as well as the timestamp from which to begin querying for event data. It is composed of at least one transaction type.
- Artifact type: The artifact type represents a document header or a document line as data objects. Each type of specialization is a subclass of artifact type. The document header class instance can have zero or more document line class instances. For every artifact type, it is necessary to specify its name and name of the table, which provides one or more fields that uniquely identify each artifact instance. There may only be one main artifact and it represents the source artifact in each relation.
- Transaction type: A transaction type only occurs for one artifact. Every transaction must have the type name and the field name of a database table that holds data about the execution time of a transaction.
- Direct relation: A direct relation describes exactly one target artifact class and one source artifact class. Besides the type name, the reference field must be specified. It relates the target artifact class to the source artifact class. Thus, the table of target artifact class provides one or more fields for reference information.
- Indirect relation: An indirect relation describes exactly one target artifact class and one source artifact class. Every indirect relation must have the name and the join path. The join path consists of two or more joins through several linked tables.

These metamodel constraints have been defined using the Object Constraint Language (OCL) [29]. This language is the de facto standard for specifying integrity constraints for both the model and metamodel levels within the UML. As an example, the OCL constraints for artifact type log instances are given in Listings 1 and 2.

Listing 1. The artifact log is composed of at least one transaction.

```
context ArtifactTypeLog
inv self.contains -> size () >=1
```

Listing 2. ArtifactSelected assigns the main artifact of the log.

```
context ArtifactTypeLog:: ArtifactSelected (mainArtifact:ArtifactType)
post: self.subject = mainArtifact
```

Furthermore, we believe it is interesting to look at the formal specification of the constraints for the direct relation element. Listing 3 specifies exactly one target artifact class and one source artifact class for each direct relation. Next, Listing 4 shows the OCL code to restrict that the main artifact represents the source artifact in each relation. These constraints are also defined for the indirect relation element.

Listing 3. The target artifact class is related to one source artifact class.

```
1: context DirectRelation
2: inv: self.source -> size () =1 and self.target -> size () =1
```

Listing 4. SourceArtifactSelected assigns the source artifact for direct relations.

```
1: context DirectRelation:: SourceArtifactSelected (mainArtifact:ArtifactType)
2: post: self.source = mainArtifact
```

It is also important to enforce that the timestamp used to begin querying for event data is older than the timestamp used to stop querying. This constraint is given in Listing 5.

Listing 5. Time scope constraint.

```
1: context ArtifactTypeLog
2: inv: self.TimestampFrom < self.TimestampTo
```

4.2. Concrete Syntax

A second design objective refers to offering the representation of the DSML, either textually or graphically, through a concrete syntax. We opted for a graphical notation since it can represent numerous relations more comprehensively and intuitively. The language elements are represented by the graphical symbols shown in Tables 1 and 2. Also, the properties and constraints for the elements are given. Each symbol has a set of properties that must be defined by the user once a symbol is added to the diagram.

Table 1. Main elements of the domain-specific modeling language.

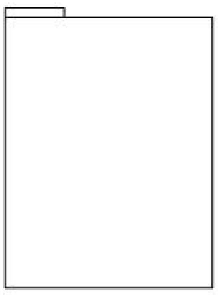
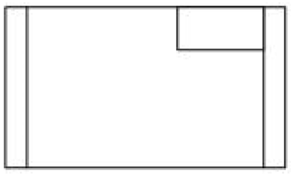
Name	Notation	Description
Artifact type log		A package is used to hold other elements in the model. It can hold as many elements as needed. The packages cannot contain other packages. A tagged value can be used to add a property to this element. The value is required and holds data about the main artifact.
Artifact type		Contains data element symbol in its upper-right corner. Symbol H represents a document header subclass, while symbol L represents a document line subclass.

Table 1. *Cont.*

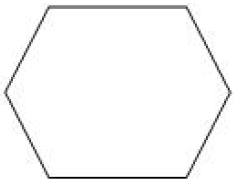
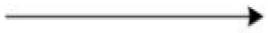
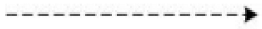
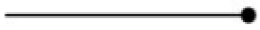
Name	Notation	Description
Transaction type		A transaction has only one package as a drop target.

Table 2. Connector elements of the domain-specific modeling language.

Name	Notation	Description
Direct relation		A solid line arrow connecting two artifacts. The arrow is drawn at the point where the line meets a target artifact. The name of a direct relation can be shown in the middle of the line. Next, reference information can be shown on the end connected to a target artifact. Both labels are optional.
Indirect relation		An indirect relation is shown as a dashed arrow line between two artifacts. The arrow is drawn at the point where the line meets a target artifact. The name of an indirect relation can be shown on the end connected to a target artifact. This label is optional.
Association		An association connects an artifact and a transaction. Association end by an artifact is indicated graphically by a small field circle. It should be interpreted as showing that transaction occurs only on one artifact.

The example in Figure 5 illustrates the graphical symbols used in the modeling notation. There is a clear distinction between language elements and each symbol has a reference to one modeling element. Explicitly, the symbol used for the artifact type log element serves as a starting point for creating a new model. It represents a composition of a set of elements that allows for reducing diagram complexity. Figure 5 shows a model where the purchase order is the main artifact.

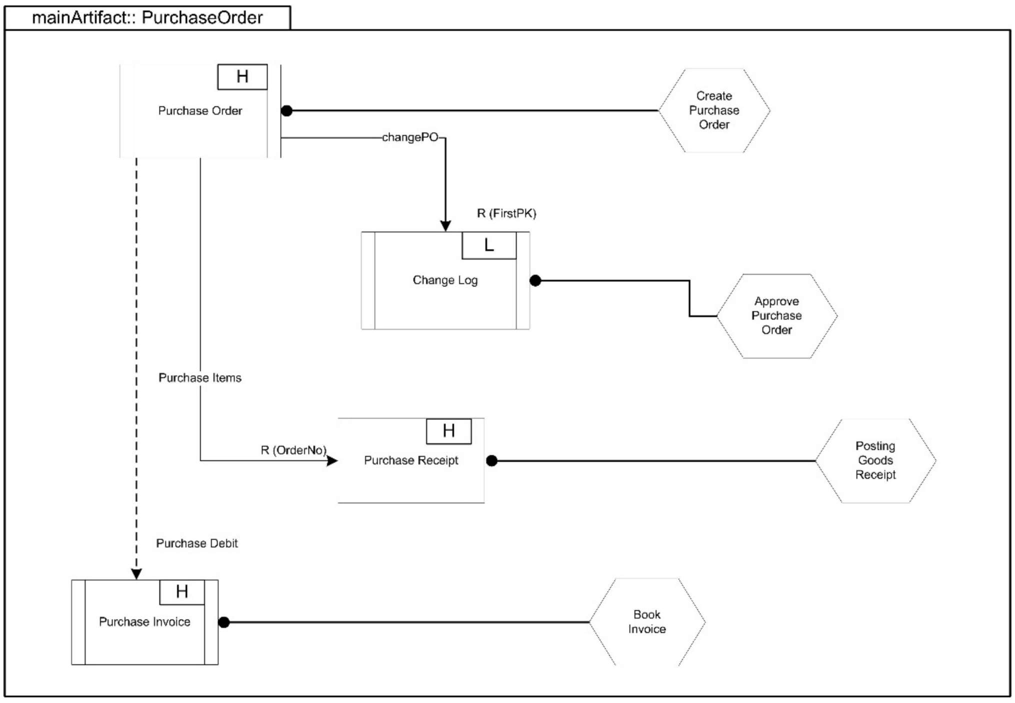


Figure 5. DSML purchase model for event log extraction.

5. Results

In this section, we demonstrate the feasibility and usefulness of the proposed DSML in a real setting. Our main motivation for developing this solution is to enable domain experts to model data of interest at a conceptual level and then automatize the extraction of event logs from ERP systems. As such, the aim of this evaluation is to verify whether the DSML model can provide sufficient information to extract and transform data from transactional ERP databases to the XES target representation. In order to validate and test the model, we use real data from a Microsoft Dynamics NAV ERP database.

5.1. Case Study

A case study has been carried out for the procurement process supported by the Microsoft Dynamics NAV ERP system. The database shares most of its elements with the running example from Section 3 and can be seen as a broader form of the running example. The data was provided by a Serbian wholesale furniture company operating in the Western Balkan region. We used NAV 2017 because the company had implemented this version of the system.

The general process behavior is given as follows. First, a new purchase order with line items is created. The purchase order needs to be approved before it can be released to a vendor. After the approval, the goods are received, and the receipt document attached to the material is inserted into the ERP system. Also, the corresponding invoice is received and registered in the system. When the invoice is posted, the details are entered into a ledger and the invoice is released for payment. Before the invoice can be paid, it is matched to one or more purchase orders. Typically, the management expects that the corresponding invoice is not paid without the registered goods receipt document. In practice, different variations of this process exist which can be explained by the fact that ERP configuration settings are not set very strict to enable the flexibility necessary in the company. Note that the purchase orders can be linked to a purchase quote or to a contract document.

Obviously, the process described above has one-to-many and many-to-many relations in the data. For instance, one invoice can be matched to multiple purchase orders and multiple invoices can be issued for a particular purchase order. Because of the existence of multiple data objects, it is difficult to choose a unique case or process instance notion for the whole process. The goal is to avoid the data convergence and data divergence problems (as explained in Section 3) by choosing proper multiple case notions or artifacts.

5.2. Step 1—Designing the Model

The first of the three main steps is designing the model for a given case. This step consists of two stages. The first stage is collecting all of the relevant inner workings and peculiarities of the system by seeking expert inputs. This leads to the second stage, which is incorporating expert opinions into a viable model.

5.2.1. Seeking Expert Inputs

The expert business opinion is used to populate elements of our metamodel. Consulting domain experts, we determine artifact types and correlate events with artifacts. Then domain knowledge is needed to obtain the relations from the data model. This is so because, for the most-used ERP systems, e.g., Microsoft Dynamics NAV ERP, the relationships between tables are not explicitly defined in the database management system (DBMS) software. Additionally, an excessive volume of data unnecessarily increases the scope of the project and consequentially complicates it. This is the reason why experts are needed. They understand the procedures and rules of the company in question, so they can outline which data are needed to optimally achieve the set goal. The experts should define the relationships between the tables, as well as main artifacts and important attributes of the database so that the event log can be properly mapped. For example, we identified two key documents in the case study, namely, the purchase order and purchase invoice. These are the primary objects in the process and an event log is obtained for each.

5.2.2. Incorporating the Expert Opinions into a Model

For the need of testing, we use the purchase order as the main artifact to validate our solution. The resulting DSML model is illustrated in Figure 5. The model includes four artifact types on different levels of granularity, namely, the purchase order, purchase receipt, purchase invoice, and change log. The change log represents the document line subclass which captures changes made to the fields on the specified document that could be translated into event types. Purchase order is defined as the main artifact of the log which is selected as a case object, i.e., a type of process instance. Different transactions can be associated with these documents, i.e., the create purchase order is linked to the purchase order artifact and the posting goods receipt is linked to the purchase receipt artifact, etc. Only the relevant transaction types should be selected considering the goal of the analysis. Further, the model consists of two direct relations and one indirect relation. The reference field is specified for each direct relation, while the join path attribute is specified as a string for purchase debit indirect relation.

5.3. Step 2—Defining SQL Query Patterns

In order to extract data, we defined SQL query patterns that will be used to create specific SQL queries whose attributes and tables would depend on the elements and properties of the model.

5.3.1. Select Main Artifact Instances

The artifact metaclass will be directly transformed into a SELECT statement. The SELECT clause specifies the identification properties of the data object to be retrieved, and the FROM clause indicates the table to be used. As stated earlier, these properties are mandatory for each artifact type. The condition metaclass attribute represents the condition to be fulfilled to distinguish various artifacts having the same table, e.g., different purchasing documents can be stored in one table. It will be transformed into a search clause. An extra condition can be used to specify the boundary date values for getting all event data in a certain timeframe. The values are mapped from timestamp attributes of artifact type log metaclass. The main artifact instances can be retrieved using the following query (see Listing 6).

Listing 6. SQL query pattern that selects main artifact instances.

```
CREATE VIEW mainArtifact AS
SELECT subjectArtifactType.FieldName_ID AS [mainArtifactID]
FROM subjectArtifactType.TableName
WHERE subjectArtifactType.Condition AND
subjectArtifactType.getFieldTimestamp() BETWEEN
ArtifactTypeLog.TimestampFrom AND ArtifactTypeLog.TimestampTo;
```

5.3.2. Obtaining Interactions between Artifacts

A list of type-level relations from the main artifact to other artifacts is obtained. Each relation class is used to generate an SQL query that joins the tables of source and target artifacts and selects the identification properties of each artifact. Specifically, direct relation instances are retrieved by joining the two tables based on one or more reference fields defined in a model. For indirect relations, it is possible to connect two tables using the join path attribute which involves multiple joins to be performed. The result is effectively a join table that holds two artifact IDs, i.e., the source and target, which allows us to identify interrelated artifacts. Because of space limitations in this paper, we illustrate one such query pattern in Listing 7.

Listing 7. SQL query pattern that selects direct relation instances.

```
SELECT DirectRelation.SourceArtifact.FieldName_ID
DirectRelation.TargetArtifact.FieldName_ID
FROM DirectRelation.SourceArtifact.TableName AS S
INNER JOIN DirectRelation.TargetArtifact.TableName AS T
ON S.DirectRelation.SourceArtifact.FieldName_ID =
T.DirectRelation.ReferenceField
WHERE DirectRelation.SourceArtifact.condition AND
DirectRelation.TargetArtifact.condition;
```

To illustrate with a concrete example in our case study, an SQL query is shown in Listing 8, which selects relation instances between purchase order and purchase receipt artifacts. Table 3 shows the excerpt of the query result. Notice that, in this case, two instances of direct relation exist between the purchase order PO1 and receipt documents R1 and R3. It should also be noted that the examples specific to the case study should not be written manually, but that they would be generated based on the SQL query patterns and the provided model.

Listing 8. SQL query that selects instances of purchase item direct relation.

```
SELECT Order_No, Receipt_No
FROM Purchase_Order_Header AS S INNER JOIN Purch_Rcpt_Header AS T
ON S.Order_No = T.OrderNo
WHERE S.Document_Type = 'Standard Purchase Order' AND
T.Document_Type = 'Purchase receipt';
```

Table 3. Excerpt of the purchase item direct relation instances.

Purchase Order (Document Number)	Purchase Receipt (Document Number)
PO1	R1
PO2	R2
PO1	R3
PO3	R4
PO3	R5
PO4	R7
PO5	R8
PO6	R9
PO7	R6
PO8	R10
PO10	R11
PO11	R12
PO12	R13

5.3.3. Select Transaction Instances

Next, the transaction metaclass, which describes the event type and timestamp, is also transformed into a SELECT statement. Each transaction type has a reference to only one artifact type according to the association relationship of the metamodel. Thus, the SELECT query has answer variables, i.e., identifiers of a related artifact that denote occurrences of transaction instances, timestamp, and identifiers of the main artifact, which establish a relation between transaction instance and process instance, i.e., the case it belongs to. The FROM clause indicates which direct or indirect relation will be used in the join to select only transactions which belong to the main artifact instances. The type-level relation between two artifacts is set up in the DSML model. For example, the purchase debit indirect relation is specified and then used to query the data concerning book invoice transactions.

We use the placeholder "<< … >>" for particular parts of the pattern which will contain the results of a subquery. The placeholder "<<relation join>>" (line 4 in Listing 9) subsumes the results of the query pattern for a type-level relation such as the one shown in Listing 8. Then, the resulting table needs to be joined with the table of the artifact to which the transaction is related, to get the timestamp of events. Additional conditions can be expressed in the where clause to distinguish transaction types or to filter the results. For example, the user can define an aggregation function to obtain only the first or last recorded transaction for the analysis. The transaction instances can be retrieved using the following query.

Listing 9. SQL query pattern that selects transaction instances.

```
1: SELECT Transaction.ArtifactTypeRef.FieldName_ID AS [event_id],
2: Transaction.FieldName_timestamp AS [event_timestamp],
3: R.mainArtifact.FieldName_ID AS [trace_ID]
4: FROM <<relation join>> AS R
5: INNER JOIN Transaction.ArtifactTypeRef.TableName AS REF
6: ON R.ArtifactTypeRef.FieldName_ID =
7: REF.Transaction.ArtifactTypeRef.FieldName_ID
8: WHERE Transaction.conditon;
```

The number of attributes and tables used in the log extraction step depends on the elements and properties of the DSML conceptual model. To create a high-level overview of the procurement process, our domain experts suggest we consider the four relevant transaction types for process analysis (see Figure 5). Therefore, the four database tables were used in the log extraction step for this case study. The Purchase Header, Purch_Rcpt_Header, Purch_Inv_Header, and Change Log Entry database tables were used.

The user is able to determine the additional constraints on which instance to consider within the conceptual model. For our analysis, we considered data recorded between 15-

06-2017 and 15-10-2017. The boundary date values for obtaining all event data in a certain timeframe are mapped from the timestamp attributes of the artifact type log metaclass. In total, 31 cases were extracted from the real data of ERP system for the main artifact Purchase Order. A fragment of the extracted event log is given in Table 4. Each line corresponds to an event.

Table 4. A fragment of the event log extracted from the real data of a NAV ERP system.

Case ID	Event ID	Activity Name	Timestamp
PO1	PO1	Create Purchase Order	20-06-2017
PO1	R1	Posting Goods Receipt	23-06-2017
PO1	R3	Posting Goods Receipt	25-06-2017
PO1	IN1	Book Invoice	23-06-2017
PO1	IN2	Book Invoice	25-06-2017
PO1	1	Approve Purchase Order	20-06-2017
PO2	PO2	Create Purchase Order	29-06-2017
PO2	R2	Posting Goods Receipt	13-07-2017
PO2	IN1	Book Invoice	13-07-2017
PO2	2	Approve Purchase Order	29-06-2017
PO3	PO3	Create Purchase Order	08-07-2017
PO3	R4	Posting Goods Receipt	10-07-2017
PO3	R5	Posting Goods Receipt	12-07-2017
PO3	IN709	Book Invoice	10-07-2017
PO3	IN710	Book Invoice	13-07-2017
PO3	3	Approve Purchase Order	08-07-2017
PO4	PO4	Create Purchase Order	13-07-2017
PO4	R7	Posting Goods Receipt	17-07-2017
PO4	IN707	Book Invoice	17-07-2017
PO5	PO5	Create Purchase Order	15-08-2017
PO5	R8	Posting Goods Receipt	25-08-2017
PO5	IN708	Book Invoice	25-08-2017
PO5	4	Approve Purchase Order	17-08-2017
PO6	PO6	Create Purchase Order	15-08-2017
PO6	R9	Posting Goods Receipt	27-09-2017
PO6	R19	Posting Goods Receipt	30-09-2017
PO6	R21	Posting Goods Receipt	01-10-2017
PO6	IN710	Book Invoice	30-09-2017
PO6	IN712	Book Invoice	01-10-2017
PO6	IN723	Book Invoice	27-09-2017
PO7	PO7	Create Purchase Order	15-08-2017
PO7	R6	Posting Goods Receipt	20-08-2017
PO7	IN713	Book Invoice	20-08-2017
PO7	5	Approve Purchase Order	17-08-2017
PO8	PO8	Create Purchase Order	21-08-2017
PO8	R10	Posting Goods Receipt	28-08-2017
PO8	IN714	Book Invoice	28-08-2017
PO8	6	Approve Purchase Order	21-08-2017
PO10	PO10	Create Purchase Order	17-08-2017
PO10	R11	Posting Goods Receipt	20-08-2017
PO10	IN715	Book Invoice	21-08-2017
PO10	7	Approve Purchase Order	17-08-2017
PO11	PO11	Create Purchase Order	24-09-2017
PO11	R12	Posting Goods Receipt	30-09-2017
PO11	IN716	Book Invoice	30-09-2017
PO11	8	Approve Purchase Order	25-09-2017
PO12	PO12	Create Purchase Order	02-10-2017
PO12	R13	Posting Goods Receipt	17-10-2017
PO12	IN731	Book Invoice	17-10-2017

5.4. Step 3—Mapping DSML Model to XES Format

We now briefly describe how to map the model provided by the user and the data provided by the queries to an XES document which can later be used for process mining. The mapping is described in Algorithm 1. Figure 6 demonstrates the excerpt of the XES document for our case study.

Algorithm 1 Mapping the DSML model into an XES document.

Input: DSML model, SQL query result sets
Output: XES document

1. Map the main artifact instances to event log case objects. The "concept:name" attribute will store the case identifier, which will allow the grouping of all events linked to the same instance into a single trace. The necessary data comes from the SQL query result set for the main artifact instances. It populates the trace element in an XES model (see Listing 10).
2. Derive the identity of each event in one of two ways:
    a. Derive the name from a descriptive name of the activity or
    b. Derive the name from a list of two attributes: the name of the activity and the event identifier.

This will make the log come with two event classifiers. Only one classifier will be used to determine the event class at a time, as well as assign a label to each event in a log. Therefore, the answer variables of queries resulting from the query pattern in Listing 9 are used in the definition of an event. Each transaction instance will be mapped to an event of related trace, i.e., the main artifact instance, according to the dependency relationship of the metamodel.

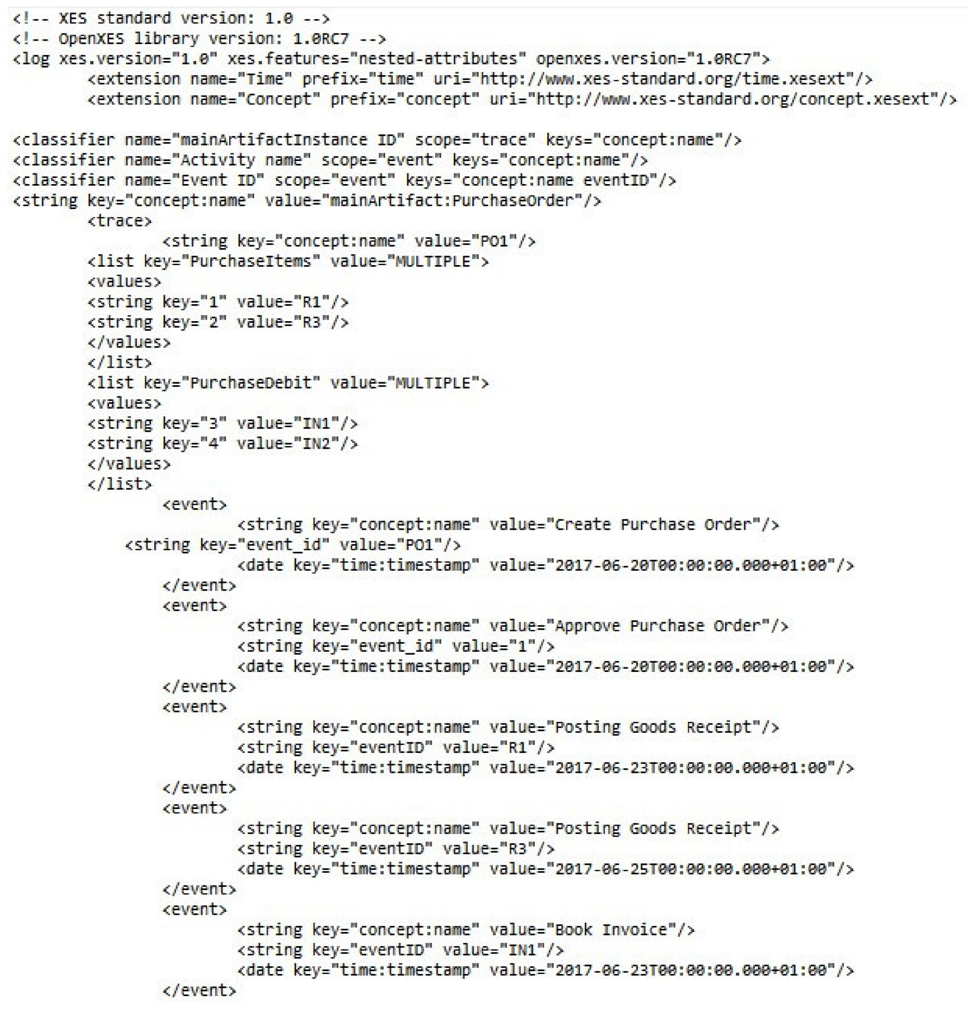
3. Define a list attribute, as a new domain-specific attribute associated with a trace, in order to specify the relationships between the main artifact and the target artifacts. One list attribute is defined for each relation type relevant to the main artifact. Use the relation name to map the list attribute key.
4. Define child attributes for each list attribute. One child attribute should correspond to one instance of the target artifact type. The child attribute is made up of a key, which is a sequence number starting from 1, and a value, which is the identifier of the target artifact. The necessary data comes from the answer variables of queries resulting from the query pattern in Listing 7. The child attribute gets its value from the target artifact identifiers as seen in Listing 11.

Listing 10. XES declaration of the name attribute for the purchase order instance.

```
<string key="concept:name" value="PO1"/>
```

Listing 11. XES declaration of the list attribute and its child attributes.

```
<list key="PurchaseItems">
<values>
<string key="1" value="R1"/>
<string key="2" value="R3"/>
</values>
</list>
```



```
<!-- XES standard version: 1.0 -->
<!-- OpenXES library version: 1.0RC7 -->
<log xes.version="1.0" xes.features="nested-attributes" openxes.version="1.0RC7">
        <extension name="Time" prefix="time" uri="http://www.xes-standard.org/time.xesext"/>
        <extension name="Concept" prefix="concept" uri="http://www.xes-standard.org/concept.xesext"/>

<classifier name="mainArtifactInstance ID" scope="trace" keys="concept:name"/>
<classifier name="Activity name" scope="event" keys="concept:name"/>
<classifier name="Event ID" scope="event" keys="concept:name eventID"/>
<string key="concept:name" value="mainArtifact:PurchaseOrder"/>
        <trace>
                <string key="concept:name" value="PO1"/>
        <list key="PurchaseItems" value="MULTIPLE">
        <values>
        <string key="1" value="R1"/>
        <string key="2" value="R3"/>
        </values>
        </list>
        <list key="PurchaseDebit" value="MULTIPLE">
        <values>
        <string key="3" value="IN1"/>
        <string key="4" value="IN2"/>
        </values>
        </list>
                <event>
                        <string key="concept:name" value="Create Purchase Order"/>
            <string key="event_id" value="PO1"/>
                        <date key="time:timestamp" value="2017-06-20T00:00:00.000+01:00"/>
                </event>
                <event>
                        <string key="concept:name" value="Approve Purchase Order"/>
                        <string key="event_id" value="1"/>
                        <date key="time:timestamp" value="2017-06-20T00:00:00.000+01:00"/>
                </event>
                <event>
                        <string key="concept:name" value="Posting Goods Receipt"/>
                        <string key="eventID" value="R1"/>
                        <date key="time:timestamp" value="2017-06-23T00:00:00.000+01:00"/>
                </event>
                <event>
                        <string key="concept:name" value="Posting Goods Receipt"/>
                        <string key="eventID" value="R3"/>
                        <date key="time:timestamp" value="2017-06-25T00:00:00.000+01:00"/>
                </event>
                <event>
                        <string key="concept:name" value="Book Invoice"/>
                        <string key="eventID" value="IN1"/>
                        <date key="time:timestamp" value="2017-06-23T00:00:00.000+01:00"/>
                </event>
```

Figure 6. A part of the XES document for the purchase order artifact.

5.5. Process Discovery

The goal of this subsection is to demonstrate that the generated XES document is valid, which would mean that the mapping is successful. We can demonstrate this by generating a business process model from the XES document with the process mining tool Disco. The resulting model can be seen in Figure 7. This shows that the XES document can be used to successfully generate process models and that the process models are in line with the previously discussed business logic, making the XES document valid. We will not analyze the model further, as it is beyond the scope of this paper.

The discovered process model represents a high-level overview of the procurement process. The language proposed in this work allows the user to choose a specific view on data for building the log for analysis. Different selections of the case notion and the relevant transactions in the conceptual design phase of our solution provide different views on data. This selection is mostly influenced by the goal of the analysis. The user can choose what parts of the event data can be interesting to look into. For our case study, one can create a different view by adding more transactions within the DSML conceptual model, so a bigger process model and different variants of the process can be discovered.

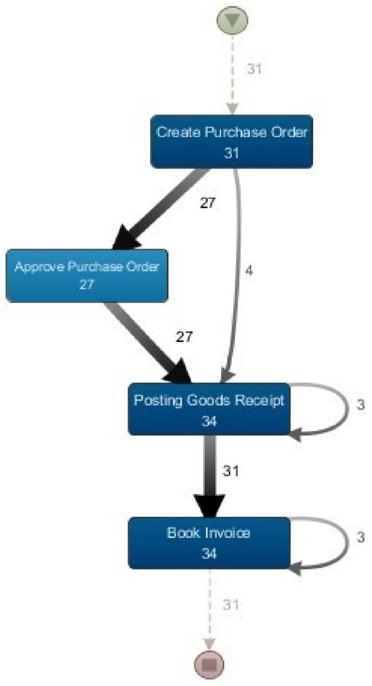


Figure 7. Discovered process model from the generated XES document.

6. Discussion

This paper has attempted to address the issue of extracting suitable event logs from data stored in relational databases, supporting domain knowledge. We have particularly focused on the business processes supported by ERP systems, which are generally seen as a collection of interrelated documents. In fact, a single process instance may involve several documents, each with its own life cycle. Hence it is very difficult to describe the evolution of various documents using a single process definition. This has led to artifact-centric approaches which can deal with one-to-many and many-to-many relations, avoiding convergence and divergence problems. These approaches have proven more appropriate for complex business processes supported by an ERP system [9].

To deal with one-to-many and many-to-many relations, the metamodel proposed in this paper provides an artifact-centric view of databases. The DSML metamodel was initially described in our previous work [30]; however, this metamodel was revised and we acknowledged some limitations compared to the new metamodel proposed in this paper. Firstly, the metamodel was incomplete in terms of its semantics and also inconsistent that made it more difficult to understand by the end-user. The artifact type log element was not considered, thus lacking properties which provide valuable information, such as the timestamp from which to begin querying for event data. Also, the attributes of metaclasses and the number of metamodel restrictions were not described. Secondly, the different levels of abstraction were within the model which compromised an appropriate interpretation of the model. For example, the trace and link elements at type level (M1) were included.

Afterward, a full specification of the language was provided in this paper. This DSML is composed of the revised metamodel with its attributes and new constraints and a graphical representation of the metamodel concepts. In addition, we have determined how certain metaclasses of language could be transformed into SQL code. This is an automatic process that does not require any further user interaction.

The case study demonstrates that it is possible to apply the developed solution to a real example. The results show that the language includes constructs that enable a domain expert to easily model data of interest in a log extraction step. Once a conceptual model has been designed, this provides sufficient information to automate the generation of queries. We have successfully described the preferred artifacts and the relations between them. We argue that our solution works in real settings because it facilitates mapping from the conceptual model and queries variables to the target XES representation. The output consists of an event log in the XES format for each main artifact. This results in avoiding the data convergence and data divergence problems, since multiple event logs are obtained with stored relations among artifacts. For this purpose, we use the list attribute in the XES standard.

We have demonstrated that the generated XES document is valid by discovering a business process model. The resulting process model is in line with the business logic presented in the case study. Furthermore, an artifact-centric process model can be discovered from these multiple event logs using existing process discovery techniques. A discussion on that topic extends beyond the scope of this paper. We refer the readers to [11,25] for more information about process discovery techniques that can be used to discover artifact life cycle models and artifact interactions.

Furthermore, the authors are aware that the main limitation of the presented solution is the lack of tool support in particular phases. In view of the positive results of the study, active work is being carried out to develop a more natural and sophisticated tool to provide better guidance to users.

7. Conclusions

In this paper, we have proposed a graphical DSML that provides support to domain experts in obtaining event logs from ERP databases. A new metamodel version with corresponding constraints has been introduced, as well as a graphical representation of the language concepts. Specifically, domain experts will be able to indicate where cases and events are located within a database at a conceptual level. Afterward, these conceptual models will be automatically validated and transformed into SQL code. When the query results are obtained, the extracted data can be mapped to XES elements. We have briefly described each component and demonstrated the applicability of the approach using a case study with real data from a Microsoft Dynamics NAV ERP database. Although the demonstration shows its usefulness, more empirical experiments are still needed. As part of our work-in-progress, we are developing a specialized CASE tool that will include a set of applications to support the implementation of each component.

Besides, the proposed language makes the querying of process data easier and domain experts can focus on modeling data of interest conceptually without the need to learn a programming language. As future work, a further formal definition of mechanisms for mapping ERP transactional data to the XES event log will be provided according to the data extraction model and transformation rules.

Author Contributions: Conceptualization, A.P.S., S.B. and O.P.; methodology, A.P.S. and S.B.; software, A.P.S. and S.K.; validation, S.B. and O.P.; investigation, A.P.S., S.B. and O.P.; writing—original draft preparation, A.P.S. and S.K.; writing—review and editing, S.B. and O.P.; visualization, A.P.S. and S.K.; supervision, S.B. and O.P. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The data presented in this study are available on request from the corresponding author. The data are not publicly available because they contain confidential business information.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Utama, N.I.; Sutrisnowati, R.A.; Kamal, I.M.; Bae, H.; Park, Y.J. Mining shift work operation from event logs. *Appl. Sci.* **2020**, *10*, 7202. [CrossRef]
2. Van der Aalst, W.M.P. *Process Mining: Data Science in Action*, 2nd ed.; Springer: Berlin/Heidelberg, Germany, 2016.

3. Van der Aalst, W.M.P. Object-Centric Process Mining: Dealing with Divergence and Convergence in Event Data. In *Software Engineering and Formal Methods (SEFM 2019). Lecture Notes in Computer Science*; Ölveczky, P., Salaün, G., Eds.; Springer: Cham, Switzerland, 2019; Volume 11724, pp. 3–25. [CrossRef]
4. Van der Aalst, W.M.P. Extracting Event Data from Databases to Unleash Process Mining. In *BPM—Driving Innovation in a Digital World. Management for Professionals*; vom Brocke, J., Schmiedel, T., Eds.; Springer: Cham, Switzerland, 2015; pp. 105–128. [CrossRef]
5. Calvanese, D.; Kalayci, T.E.; Montali, M.; Santoso, A. OBDA for log extraction in process mining. In *Reasoning Web. Semantic Interoperability on the Web. Reasoning Web 2017. Lecture Notes in Computer Science*; Ianni, G., Lembo, D., Bertossi, L., Faber, W., Glimm, B., Gottlob, G., Staab, S., Eds.; Springer: Cham, Switzerland, 2017; Volume 10370, pp. 292–345. [CrossRef]
6. Dakic, D.; Stefanovic, D.; Lolic, T.; Narandzic, D.; Simeunovic, N. Event Log Extraction for the Purpose of Process Mining: A Systematic Literature Review. In *Innovation in Sustainable Management and Entrepreneurship (SIM 2019). Springer Proceedings in Business and Economics*; Prostean, G., Lavios Villahoz, J., Brancu, L., Bakacsi, G., Eds.; Springer: Cham, Switzerland, 2020; pp. 299–312. [CrossRef]
7. Jans, M.; Soffer, P.; Jouck, T. Building a valuable event log for process mining: An experimental exploration of a guided process. *Enterp. Inf. Syst.* **2019**, *13*, 601–630. [CrossRef]
8. Erdogan, T.G.; Tarhan, A. A goal-driven evaluation method based on process mining for healthcare processes. *Appl. Sci.* **2018**, *8*, 894. [CrossRef]
9. Fahland, D. Artifact-Centric Process Mining. In *Encyclopedia of Big Data Technologies*; Sakr, S., Zomaya, A.Y., Eds.; Springer: Cham, Switzerland, 2019; pp. 1–13. [CrossRef]
10. IEEE Standard for eXtensible Event Stream (XES) for Achieving Interoperability in Event Logs and Event Streams. *IEEE Std 1849–2016*; The Institute of Electrical and Electronics Engineers: New York, NY, USA, 2016; pp. 1–50. [CrossRef]
11. Lu, X.; Nagelkerke, M.; van de Wiel, D.; Fahland, D. Discovering Interacting Artifacts from ERP Systems. *IEEE Trans. Serv. Comput.* **2015**, *8*, 861–873. [CrossRef]
12. Wieczorek, S.; Stefanescu, A.; Schieferdecker, I.K. Test data provision for ERP systems. In Proceedings of the 2008 1st International Conference on Software Testing, Verification, and Validation, Lillehammer, Norway, 9–11 April 2008. [CrossRef]
13. De Murillas, E.G.L.; Reijers, H.A.; van der Aalst, W.M.P. Connecting databases with process mining: A meta model and toolset. *Softw. Syst. Model.* **2019**, *18*, 1209–1247. [CrossRef]
14. Jans, M.; Soffer, P. From Relational Database to Event Log: Decisions with Quality Impact. In *Business Process Management Workshops (BPM 2017). Lecture Notes in Business Information Processing*; Teniente, E., Weidlich, M., Eds.; Springer: Cham, Switzerland, 2018; Volume 308. [CrossRef]
15. Diba, K.; Batoulis, K.; Weidlich, M.; Weske, M. Extraction, correlation, and abstraction of event data for process mining. *WIREs Data Min. Knowl. Discov.* **2020**, *10*. [CrossRef]
16. Nooijen, E.H.J.; van Dongen, B.F.; Fahland, D. Automatic Discovery of Data-Centric and Artifact-Centric Processes. In *Business Process Management Workshops (BPM 2012). Lecture Notes in Business Information Processing*; La Rosa, M., Soffer, P., Eds.; Springer: Berlin/Heidelberg, Germany, 2013; Volume 132, pp. 316–327. [CrossRef]
17. 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 *Information Systems in the Big Data Era (CAiSE) 2018. Lecture Notes in Business Information Processing*; Mendling, J., Mouratidis, H., Eds.; Springer: Cham, Switzerland, 2018; Volume 317, pp. 182–199. [CrossRef]
18. Li, G.; de Carvalho, R.M.; van der Aalst, W.M.P. Object-Centric Behavioral Constraint Models: A Hybrid Model for Behavioral and Data Perspectives. In Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing (SAC '19), Limassol, Cyprus, 8–12 April 2019.
19. Li, G.; de Carvalho, R.M.; van der Aalst, W.M.P. Automatic Discovery of Object-Centric Behavioral Constraint Models. In *Business Information Systems (BIS 2017). Lecture Notes in Business Information Processing*; Abramowicz, W., Ed.; Springer: Cham, Switzerland, 2017; Volume 288, pp. 43–58. [CrossRef]
20. Verbeek, H.M.W.; Buijs, J.C.A.M.; van Dongen, B.F.; van der Aalst, W.M.P. XES, XESame, and ProM 6. In *International Conference on Advanced Information Systems Engineering (CAiSE 2010). Lecture Notes in Business Information Processing*; Soffer, P., Proper, E., Eds.; Springer: Berlin/Heidelberg, Germany, 2010; Volume 72, pp. 60–75. [CrossRef]
21. Günther, C.W.; van der Aalst, W.M.P. A Generic Import Framework For Process Event Logs. In *Business Process Management Workshops. BPM 2006. Lecture Notes in Computer Science*; Eder, J., Dustdar, S., Eds.; Springer: Berlin/Heidelberg, Germany, 2006; Volume 4103, pp. 81–92. [CrossRef]
22. Ingvaldsen, J.E.; Gulla, J.A. Preprocessing Support for Large Scale Process Mining of SAP Transactions. In *Business Process Management Workshops (BPM 2007). Lecture Notes in Computer Science*; ter Hofstede, A., Benatallah, B., Paik, H.Y., Eds.; Springer: Berlin/Heidelberg, Germany, 2008; Volume 4928, pp. 30–41. [CrossRef]
23. Piessens, D. Event Log Extraction from SAP ECC 6.0. Master's Thesis, Eindhoven University of Technology, Eindhoven, The Netherlands, 2011.
24. De Murillas, E.G.L.; van der Aalst, W.M.P.; Reijers, H.A. Process Mining on Databases: Unearthing Historical Data from Redo Logs. In *Business Process Management Workshops (BPM 2016). Lecture Notes in Computer Science*; Motahari-Nezhad, H., Recker, J., Weidlich, M., Eds.; Springer: Cham, Switzerland, 2015; Volume 9253, pp. 367–385. [CrossRef]
25. Popova, V.; Fahland, D.; Dumas, M. Artifact lifecycle discovery. *Int. J. Coop. Inf. Syst.* **2015**, *24*, 1–27. [CrossRef]

26. Magal, S.R.; Word, J. *Integrated Business Processes with ERP Systems*, 1st ed.; Wiley Publishing: Hoboken, NJ, USA, 2011.
27. Frank, U. Domain-Specific Modeling Languages: Requirements Analysis and Design Guidelines. In *Domain Engineering*; Reinhartz-Berger, I., Sturm, A., Clark, T., Cohen, S., Bettin, J., Eds.; Springer: Berlin/Heidelberg, Germany, 2013; pp. 133–157. [CrossRef]
28. Object Management Group (OMG). *OMG Unified Modeling Language (UML)*, version 2.5; 2015. Available online: https://www.omg.org/spec/UML/About-UML/ (accessed on 12 March 2021).
29. Object Management Group (OMG). *Meta Object Facility (MOF) Core Specification*, version 2.5.1; 2016. Available online: https://www.omg.org/spec/MOF/2.5.1/PDF (accessed on 5 June 2021).
30. Pajić Simović, A.; Babarogić, S.; Pantelić, O. A Domain-Specific Language for Supporting Event Log Extraction from ERP systems. In Proceedings of the 2018 7th International Conference on Computers Communications and Control (ICCCC), Oradea, Romania, 8–12 May 2018.