ABSTRACT Process discovery techniques can automatically discover process models from event data. These models reveal the actual behavior of the related organization and have successfully been applied in a range of domains. Event data need to be extracted from information systems. Today, most organizations use object-centric systems such as ERP and CRM systems, which generate and store data in an object-centric manner. Unfortunately, existing discovery techniques are more focused on a behavioral perspective of processes, where the data perspective is often considered as a second-class citizen. Moreover, these discovery techniques fail to deal with object-centric data with many-to-many relationships. Event data need to be "flattened" focusing on a single object type (i.e., case notion). Therefore, in this paper, we aim to discover a novel model which combines data and behavior perspectives, and the resulting Object-Centric Behavioral Constraint (OCBC) model is able to describe processes involving interacting instances and complex data dependencies. Besides, we provide solutions to deal with the noise problem, which enables process discovery in real life data.

INDEX TERMS Process mining, process discovery, cardinality constraints, OCBC models, noise.

I. INTRODUCTION
Information systems are widely used by enterprises and organizations to support their daily transactions. Object-centric systems such as ERP and CRM, are the most widely used type of systems used by today’s organizations. These systems generate and store data in an object-centric manner, i.e., transactions update a database (often relational) storing information about objects (people, orders, etc.). Such systems integrate and optimize the business processes and transactions over different departments using shared databases maintained by a database management system [1], [3], [23].

Process discovery techniques can automatically discover process models from event data generated by information systems. These models reveal the actual behavior of the related organization and have successfully been applied in a range of domains. The existing state of the art techniques such as the Inductive Miner [16], ILP Miner [34], Heuristic Miner [36], Declare Miner [21] and Split Miner [2] assume that process instances are recorded as cases composed of ordered events. In this classical setting each event corresponds to one case identifier [30].

However, most of these discovery techniques fail to deal with data-centric/object-centric processes supported by CRM and ERP systems, which have one-to-many and many-to-many relationships between data objects. The immediate challenge for existing techniques is that these processes have no clear case notion to relate events. Besides, the discovered models are often based on business process modeling languages such as Petri nets, BPMN diagrams, Workflow nets, EPCs, and UML activity diagrams, which typically consider process instances in isolation, ignoring interactions among them. Moreover, they cannot model the data perspective in a precise manner. In other words, the data perspective can be described, but the more powerful constructs (e.g., cardinality constraints) used in Entity-Relationship (ER) models [4], Object-Role Models (ORM) [11] and UML class models [10] are not employed at all. As a result, data and control-flow need to be described in separate diagrams.
Researchers have proposed different ways to solve the problems mentioned above. The early solutions extend process models with a data perspective, while remaining process-centric. For instance, [8], [9], [13], [14], [37] append values to tokens in Petri nets in order to add data to process models, resulting in colored Petri nets. Several variants of colored Petri nets tailored towards process mining and BPM have been proposed. Data-aware process mining discovery techniques [7], [28] extend the control-flow perspective with the data perspective (e.g., read and write operations, decision points and transition guards) using standard data mining techniques. These techniques can express the data perspective to some extent, but are still not powerful enough. Besides, they are based on logs with case notions. The latter solutions are artifact-centric approaches proposed in [5], [12], [18], [24], and [31], which attempt to describe business processes in terms of so-called business artifacts. Artifacts have data and lifecycles attached to them, thus relating both perspectives. References [19], [25], and [27] demonstrate several approaches to discover artifact-centric models from data-centric processes. However, they force users to specify artifacts as well as a single instance notion within each artifact, and tend to result in complex specifications over multiple diagrams.

This paper aims to discover models based on a novel modeling language, named the Object-Centric Behavioral Constraint (OCBC) language [32], that combines data/object modeling techniques (ER, UML, or ORM) and a declarative process modeling language (inspired by Declare [33]). More precisely, a discovered OCBC model (cf. Figure 10) consists of a class model (presenting cardinality constraints between objects), a behavioral model (presenting declarative constraints between events) and so-called AOC relationships which connect these two models by relating activities in the behavioral model to object classes in the class model. Besides, we provide solutions to simplify OCBC models by filtering infrequent elements and discover precise behavioral constraints, which can better discover models in real life data with noise than the approach in [17].

The remainder is organized as follows. Section II proposes a novel log format to organize object-centric data displayed above, which is taken as input for model discovery techniques. Our discovery algorithm is proposed in Section III while illustrating the ingredients of OCBC models. Section IV provides advanced discovery techniques to deal with noise. We implement the discovery approach in a ProM plugin, which is described in Section V. The plugin is interactive and allows the user to customize the process model. Section VI compares OCBC models with other common models and Section VII concludes the paper.

II. OBJECT-CENTRIC EVENT LOGS

The input for process discovery techniques is an event log, which contains the event data generated by information systems. The XES log format which stands for eXtensible Event Stream (www.xes-standard.org) is the de facto exchange
format for discovery techniques [35]. In general, an XES log consists of a collection of traces. A trace describes the life-cycle of a particular case (i.e., a process instance) in terms of the activities executed. When the data have many-to-many relationships or multiple case notions, we need to flatten the data to fit the single case paradigm required by the XES log format. The flattened event log is like a particular view on the whole data set and it ruins the completeness of the original data set.

As object-centric data have many-to-many relationships, existing event log formats fail to organize such data (without “flattening” the data). In order to keep the completeness, we propose a novel object-centric event log format to organize and store object-centric data.

A. OBJECTS AND CLASSES

Figure 1 displays a motivating example which is used to illustrate our approach throughout the paper. The example is a fragment of object-centric data, generated in the Order-To-Cash (OTC) business process scenario of a real ERP system named Dolibarr. In general, the data consists of records in 9 tables, which correspond to transactions on Dolibarr. For instance, the first row $o1$ in the “order” table indicates an order was created at “2017-08-11 10:33:37”. The directed edges between tables indicate the dependency relations. For instance, $r5$ denotes a dependency relation from the “customer” column (foreign key) in the “order” table to the “id” column (primary key) in the “customer” table. Based on this relation, we know that the order $o1$ is for a customer $c1$ named “Ming”. Note that, there exist one-to-many relations, e.g., one order $o1$ corresponds to two order lines $ol1$ and $ol2$ in the “order_line” table. Besides, there also exist many-to-many relations illustrated by the “element relation” table, in which, one order $o1$ corresponds to two invoices $i1$ and $i2$ while one invoice $i2$ corresponds to two orders $o1$ and $o2$.

The data shown in Figure 1 are object-centric, which are different from process-centric data such as XES logs. The object-centric manner is the way data are organized in relational databases (SAP, Oracle, etc.). The term “object” in this paper has a different meaning from other fields, such as software engineering. Objects are data elements generated by object-centric information systems and stored in a relational database. These are grouped in classes and have some attributes. Besides, there exist object relations between objects to indicate the interrelation. An object relation is of some class relationship, which indicates the type of interrelation.

For example in Figure 1, the record $o1$ in the “order” table can be considered as an object of class “order”. Each value (e.g., “c1” in the “customer” column) in the record can be considered as an attribute of the object. Class relationships correspond to dependency relationships between tables, which are directed (denoted as arrows) from child tables to parent tables. For instance, the class relationship $r5$ indicates the foreign key “customer” in the “order” table refers to the primary key “id” in the “customer” table. An object relation example of $r5$ is the relation between $c1$ and $o1$, which indicates the customer of $o1$ is $c1$.

Objects can represent records in tables while object relations can represent the reference relations between records. Based on this idea, we define an object model to incorporate objects and object relations to represent the database.

**Definition 1 (Object Model):** Let $\mathcal{U}_O$ be the universe of object identifiers, $\mathcal{U}_C$ be the universe of classes, $\mathcal{U}_R$ be the universe of class relationships, $\mathcal{U}_{Att}$ be the universe of attribute names and $\mathcal{U}_{Val}$ be the universe of attribute values. An object model is a tuple $\mathcal{OM} = (\mathcal{Obj}, \mathcal{Rel}, \mathcal{Class}, \mathcal{Att})$, where:

- $\mathcal{Obj} \subseteq \mathcal{U}_O$ is the set of objects,
- $\mathcal{Rel} \subseteq \mathcal{U}_R \times \mathcal{Obj} \times \mathcal{Obj}$ is the set of object relations, and
- $\mathcal{Class} \subseteq \mathcal{Obj} \rightarrow \mathcal{U}_C$ maps objects onto classes.
- $\mathcal{Att} \subseteq \mathcal{Obj} \rightarrow (\mathcal{U}_{Att} \neq \mathcal{U}_{Val})$ maps objects onto a partial function assigning values to some attributes.\(^2\)

$\mathcal{U}_{OM}$ is the universe of object models.

Figure 2 shows an object model example $\mathcal{OM} = (\mathcal{Obj}, \mathcal{Rel}, \mathcal{Class}, \mathcal{Att})$. Note that objects are represented by unique identifiers which enable us to refer to a specific object. Even two objects with the same properties are still distinguishable by their identity. The objects are depicted as grey dots: $\mathcal{Obj} = \{o1, o2, ol1, \ldots, ol4, c1, c2, erl, \ldots, er4, s1, s2, s3, sll, \ldots, sl5, pl1, pl2, pl3, i1, i2, i3\}$. There are two objects belonging to object class order (denoted as o), i.e., $\mathcal{Class}(o1) = \mathcal{Class}(o2) = \mathcal{Order}$. There are three object relations corresponding to class relationship $r1$, e.g., $(r1, c1, i1) \in \mathcal{Rel}$, $(r1, c1, i2) \in \mathcal{Rel}$ and $(r1, c1, i3) \in \mathcal{Rel}$. Besides, objects can also have additional attribute names and values. For instance, $i1$ may have (not shown) an attribute named “creation_date” with a value “2017-08-15 09:13:27”, i.e., $\mathcal{Att}(i1)(\text{"creation_date"}) = \text{"2017 - 08 - 15 09 : 13 : 27"}$.

\(^{2}f \in X \rightarrow Y\) is a partial function with domain $\text{dom}(f) \subseteq X$.\(^1\)

\(^1\)Dolibarr ERP/CRM is an open source (webpage-based) software package for small and medium companies (www.dolibarr.org). It supports sales, orders, procurement, shipping, payments, contracts, project management, etc.

\(^2\)
An object model consists of objects (with attributes) and corresponding object relations. It is a snapshot of the database at some moment. However, the application of object models is not limited to the field of databases. The basic idea of object models is to represent a state of an object-centric system at some moment.

B. EVENT LOGS

In general, event logs are used to record past events related to information systems, machines, organizations, etc. In this paper, the idea of event logs is to record operations on object-centric systems and the corresponding state of the system after each operation. An event log consists of a list of events which represents operations, and each event corresponds to an object model which represents the state of the process just after the execution of the event.

In object-centric data, events are recorded implicitly. For example, they could be the instances of timestamp columns. Each event corresponds to an activity and may have attributes such as the time at which the event took place, and the resource executing the corresponding event. Moreover, events are atomic and ordered. For simplicity, we assume a total order. To model the overlapping of activities in time one can use start and complete events.

Take Figure 1 as an example to explain events and activities. The “creation date” column in the “order” table can be considered as the “create order” activity while each instance of the column is a “create order” event. The additional attributes of events could be some other related columns in the same table while the total order could be achieved by timestamp or logic knowledge. Definition 2 formalizes the notion of an event log.

Definition 2 (eXtensible Object-Centric Event Log): Let \( \mathcal{U}_E \) be the universe of events and \( \mathcal{U}_A \) be the universe of activities. An eXtensible Object-Centric (XOC) event log is a tuple \( L = (E, act, attrE, EO, om, \preceq) \), where:

- \( E \subseteq \mathcal{U}_E \) is a set of events,
- \( act \in E \rightarrow \mathcal{U}_A \) maps events onto activities,
- \( attrE \in E \rightarrow (\mathcal{U}_{Attr} \uplus \mathcal{U}_{id}) \) maps events onto a partial function assigning values to some attributes,
- \( EO \subseteq E \times \mathcal{U}_O \) relates events and object references,
- \( om \in E \rightarrow \mathcal{U}_{OM} \) maps each event to the object model directly after the event took place, and
- \( \preceq \subseteq E \times E \) defines a total order on events.

\( \mathcal{U}_L \) is the universe of XOC event logs.

In the context of an XOC event log \( L \), each event \( e \) is associated with object model \( OM_e = (Obj_e, Rel_e, class_e, attrO_e) = om(e) \). In this paper, we refer directly to \( Obj_e, Rel_e, class_e, attrO_e \) for \( e \in E \) if the context is clear. Moreover, we assume objects cannot change classes in the whole log. Definition 3 defines notations that refer to events following some temporal restriction specified by \( \preceq \).

\[ \begin{align*}
\preceq_1(E) &= \{ e' \in E \mid e' \preceq e \} \text{ are the events before and including } e, \\
\preceq_2(E) &= \{ e' \in E \mid e \preceq e' \} \text{ are the events after and including } e, \\
\preceq_3(E) &= \{ e' \in E \mid e' < e \} \text{ are the events before } e,^4 \\
\triangledown_1(E) &= \{ e' \in E \mid e < e' \} \text{ are the events after } e, \text{ and} \\
\partial_e(E) &= \{ e' \in E \mid act(e') = a \} \text{ are the events corresponding to activity } a \in \mathcal{U}_A.
\end{align*} \]

An XOC event log is a collection of events that belong together, i.e., they belong to some “process” where many types of objects/instances may interact. Each event corresponds to an object model and the event refers to objects in the object model. Note that one event may refer to multiple objects and one object may be referred to by multiple events. The objects referred to by an event indicate that they are impacted by the operation corresponding to the event. Such an event log is object-centric since the events are related through the data perspective (cf. Section III-C1).

Table 1 presents an XOC log example which originates from the nine tables in the motivating example (Figure 1). More precisely, the numbers in the “Index” column indicate the order of events (i.e., \( \preceq \)). In order to refer to a specific event, the “Event” column assigns an ID to each event, e.g., \( co1 \) refers to the first create order event. Each event has a corresponding activity indicated by the “Activity” column. For instance, the activity of \( co1 \) is \( co \) (denoting create order). Besides, each event may have additional attributes. For example, the last event \( cp2 \) has two attributes, “create time = 2017-08-16” (we only show the date for simplicity), and “amount = 1904”. In order to relate the behavioral perspective and the data perspective (i.e., events and objects), each event refers to at least one object indicated by the “Reference” column, e.g., \( ol1 \) refers to \( ol \) and \( er1 \). Moreover, each event has a corresponding object model consisting of objects and object relations (in the “Object Model” column), which represents the state of the process just after the execution of the event. For instance, the object model in the first row indicates the state of the database (i.e., there are two records in the “customer” table, one record in the “order” table and two records in the “order line” table) after the first create order event (\( co1 \)) happens.

Note that, each object has a corresponding class and additional attributes. In the table, the class is indicated by its ID while the additional attributes are omitted. For instance, \( ol1 \) indicates its corresponding class is \( ol \). Table 1 also illustrates the evolution of object models. After the occurrence of some event, objects may have been added, and relations may have been added or removed.

---

3 A total order is a binary relation that is (1) antisymmetric, i.e. \( e_1 \preceq e_2 \) and \( e_2 \preceq e_1 \) implies \( e_1 = e_2 \), (2) transitive, i.e. \( e_1 \preceq e_2 \) and \( e_2 \preceq e_3 \) implies \( e_1 \preceq e_3 \), and (3) total, i.e. \( e_1 \preceq e_2 \) or \( e_2 \preceq e_1 \).

4 \( e' < e \) if and only if \( e' \preceq e \) and \( e' \neq e \).
III. DISCOVERY OF OBJECT-CENTRIC BEHAVIORAL MODELS

In this section, we propose a novel algorithm different from the one proposed in [17] to discover OCBC models from XOC logs with noise.

A. DISCOVERY OF CLASS MODELS

As OCBC models are object-centric, i.e., events are correlated by the data perspective, we discover the data perspective (i.e., class models) first. A class model defines a "space" of possible object models, i.e., concrete collections of objects and relations instantiating the class model. Therefore, the class model is discovered based on object models.

Each object has a corresponding class, e.g., object o1 is of class c1, and these classes form the framework of a class model. Besides, objects may have a number of related objects and the number has to follow some restrictions required by the business process. For instance in the OTC scenario, each order object should have precisely one related customer object. Accordingly, the discovered class model should have relationships between classes to indicate these constraints.

Considering the above idea, we need to choose or propose a modeling language to formalize class models. After reviewing all existing class (data and object) modeling languages (such as ER, ORM, UML, etc.), we extend a subset of UML class model notations as our class modeling language. More precisely, our relationships extend cardinalities with temporal annotations to represent the constraints. Cardinalities employ non-empty sets of integers (e.g., \{0, 1\} is a cardinality containing two integers) to specify the constraints on the number perspective. Temporal annotations categorize cardinalities into "eventually" cardinalities (indicated by ♦) and "always" cardinalities (indicated by □). "always" indicates that the constraint should hold at any point in time while "eventually" indicates that the constraint should hold from some point onwards.

**Definition 4 (Cardinalities):**

\[ \mathcal{U}_{\text{Card}} = \{ X \subseteq \mathbb{N} | X \neq \emptyset \} \]

defines the universe of all possible cardinalities. Elements of \( \mathcal{U}_{\text{Card}} \) specify non-empty sets of integers.
Based on the discussion above, the following definition describes the discovery algorithm for class models, which serves as the backbone for OCBC models.

**Definition 5 (Discovery of Class Model):** Let \( L = (E, \text{act}, \text{attr}E, EO, \text{om}, \leq) \) be an XOC log. The discovered class model from \( L \) is a tuple \( \text{ClaM} = (\text{OC}, \text{RT}, \pi_1, \pi_2, \pi_{\text{src}}, \pi_{\text{tar}}, \pi'_{\text{src}}, \pi'_{\text{tar}}) \), where:

- \( \text{OC} \subseteq \mathcal{U}_{\text{OC}} \) is a set of object classes such that \( \text{OC} = \{ o \mid \exists e \in E, o \in \text{Obj}_e : \text{class}(o) = \text{oc} \} \),
- \( \text{RT} \subseteq \mathcal{U}_{\text{RT}} \) is a set of relationship types (\( \text{OC} \cap \text{RT} = \emptyset \) such that \( \text{RT} = \{ r \mid \exists e \in E, o_1, o_2 \in \text{Obj}_e : (r, o_1, o_2) \in \text{Rel}_e \} \),
- \( \pi_1 \in \text{RT} \rightarrow \text{OC} \) gives the source of a relationship such that \( \forall r \in \text{RT} : (\exists e \in E, (r, o_1, o_2) \in \text{Rel}_e : \pi_1(r) = \text{class}(o_1)) \),
- \( \pi_2 \in \text{RT} \rightarrow \text{OC} \) gives the target of a relationship such that \( \forall r \in \text{RT} : (\exists e \in E, (r, o_1, o_2) \in \text{Rel}_e : \pi_2(r) = \text{class}(o_2)) \),
- \( \pi_{\text{src}} \in \text{RT} \rightarrow \mathcal{U}_{\text{Card}} \) gives the source “always” cardinality of a relationship such that \( \forall r \in \text{RT} : \pi_{\text{src}}(r) = [n \mid \exists e \in E, o_2 \in \partial_{\pi_2(e)}(\text{Obj}_e) : n = |\{ (o_1, r, o_2) \mid (r, o_1, o_2) \in \text{Rel}_e \}] \),
- \( \pi_{\text{tar}} \in \text{RT} \rightarrow \mathcal{U}_{\text{Card}} \) gives the target “always” cardinality of a relationship such that \( \forall r \in \text{RT} : \pi_{\text{tar}}(r) = [n \mid \exists e \in E, o_1 \in \partial_{\pi_1(e)}(\text{Obj}_e) : n = |\{ (o_1, r, o_2) \mid (r, o_1, o_2) \in \text{Rel}_e \}] \),
- \( \pi'_{\text{src}} \in \text{RT} \rightarrow \mathcal{U}_{\text{Card}} \) gives the source “eventually” cardinality of a relationship such that \( \forall r \in \text{RT} : \pi'_{\text{src}}(r) = [n \mid \exists e \in E, o_2 \in \partial_{\pi_2(e)}(\text{Obj}_e) : n = |\{ (o_1, r, o_2) \mid (r, o_1, o_2) \in \text{Rel}_e \}] \), where \( e_l \) is the last event in \( L \),
- \( \pi'_{\text{tar}} \in \text{RT} \rightarrow \mathcal{U}_{\text{Card}} \) gives the target “eventually” cardinality of a relationship such that \( \forall r \in \text{RT} : \pi'_{\text{tar}}(r) = [n \mid \exists e \in E, o_1 \in \partial_{\pi_1(e)}(\text{Obj}_e) : n = |\{ (o_1, r, o_2) \mid (r, o_1, o_2) \in \text{Rel}_e \}] \), where \( e_l \) is the last event in \( L \).

\( \mathcal{U}_{\text{ClaM}} \) is the universe of class models.

Figure 3 presents a class model example which contains 9 object classes and 10 relationship types, i.e., \( \text{OC} = \{ \text{customer}, \text{order}, \text{order line}, \text{shipment line}, \text{invoice}, \text{element relation}, \text{payment}, \text{payment line} \} \) and \( \text{RT} = \{ r_1, r_2, \ldots, r_{10} \} \). Each node represents one object class and each edge represents a relationship type. \( \pi_1 \) and \( \pi_2 \) are used to refer to the source class and target class of a relationship, respectively. For instance, \( \pi_1(r_1) = \text{customer} \) and \( \pi_2(r_1) = \text{invoice} \). In order to distinguish source and target sides in the graph, we add an arrow on each edge from the target side to the source side (similar to the reference relations between tables). The function \( \partial_{\text{src}} \) is indicated by all the \( \square \) cardinalities on the source side of each relationship type, e.g., \( \partial_{\text{src}}(r_1) = \{1\}, \partial_{\text{src}}(r_2) = \{1\}, \partial_{\text{src}}(r_{10}) = \{1\} \), \( \partial'_{\text{src}}, \partial_{\text{tar}}, \partial'_{\text{tar}} \) and \( \pi_{\text{src}}, \pi_{\text{tar}} \) are shown in the graph in the same way.

To illustrate the discovery technique presented in Definition 5, the following steps show how the model in Figure 3 is discovered from the example log in Table 1:

- \( \text{OC} \) can be learned by incorporating all classes of all objects in the object models of all events. For instance, \( \text{customer} \) is a discovered class since object models contain objects of class \( \text{customer} \), e.g., c1 in the object model of the event \( \text{co1} \) (c1 is an object of class c, denoting \( \text{customer} \) for short).
- \( \text{RT} \), \( \pi_1 \) and \( \pi_2 \) can be learned by observing object relations in object models of all events. For example, \( r_5 \) is discovered because there exist object relations involving \( r_5 \) in object models, e.g., \( (r_5, c1, o1) \). Besides, \( \pi_1(r_5) = \text{customer} \) and \( \pi_2(r_5) = \text{order} \) since in \( (r_5, c1, o1) \), \( c1 \) is of class \( \text{customer} \) and \( o1 \) is of class \( \text{order} \).
- For each relationship \( r \), \( \partial_{\text{src}}(r) (\partial_{\text{tar}}(r)) \) can be derived by integrating the number of object relations corresponding to \( r \) of each target (source) object in the object model of each event.\(^7\) For instance, \( \partial_{\text{src}}(r_5) (\partial_{\text{tar}}(r_5)) \) is \{1\} since in the object model of each event, each \( \text{order} \) (i.e., the target class of \( r_5 \) object (i.e., \( o1 \) and \( o2 \) has precisely one related \( \text{customer} \) object \( c1 \).\(^5\)

In contrast, \( \partial'_{\text{src}}(r_9) (\partial'_{\text{tar}}(r_9)) \) can be determined by only observing the last event in the log, i.e., incorporating the number of object relations corresponding to \( r \) of each target (source) object in the last object model. For example, \( \partial'_{\text{src}}(r_9) = \{1, 2\} \), since in the object model of the last event (i.e., \( cp2 \)), each of \( o2, o3 \) and \( o4 \) has one related \( \text{shipment line} \) object (i.e., \( sl3, sl4 \) and \( sl5 \), respectively) while \( o1 \) has two related \( \text{shipment line} \) objects (i.e., \( sl1 \) and \( sl2 \)).

In this paper, we only discover the normalized cardinalities, i.e., \( \{1\}, \{0,1\}, \{0,1\} \) and \( \{1,1\} \) (denoted as “1”, “0..1”, “*” and “1..1” respectively in the graph). In other words, the directly discovered cardinalities are transformed into the normalized cardinalities. For instance, \( \{1, 2\} \) (i.e., \( \partial'_{\text{src}}(r_9) \)) is normalized to \( \{1*\} \). The normalization extends the discovered cardinalities and allows more behavior in the discovered models which may not be observed in the log. It will decrease the precision of the models, but increase their generalization. As future work, we propose to discover other cardinalities such as “1..k”.

**B. DISCOVERY OF AOC RELATIONSHIPS**

The main advantage of an OCBC model is that it can present data and behavioral perspectives in a single diagram.

---

\(^5\)We assume that all the first (second) objects of object relations corresponding to one class relationship are of the same class in an XOC log, i.e., \( \forall e, e' \in E : (r, o_1, o_2) \in \text{Rel}_e, (r', o_1', o_2') \in \text{Rel}_{e'} : r = r' \Rightarrow \text{class}(o_1) = \text{class}(o_1'), \text{class}(o_2) = \text{class}(o_2') \).

\(^6\)\( \text{class}(o) = \{ o \mid \text{Obj}_e \land \text{class}(o) = \text{oc} \} \) denotes the whole set of objects in class oc.

\(^7\)An object \( o \) is a target (source) object of a relationship \( r \) in an event \( e \) if \( \text{class}(o) = \pi_2(r) (\text{class}(o) = \pi_1(r)) \).
In Section III-A, we defined and discovered the data perspective, i.e., a class model. The subsequent tasks are defining and discovering the behavioral perspective, i.e., behavioral constraints between activities. Different from existing discovery approaches based on XES logs, and due to the deliberate lack of case notion in XOC logs, events should be somehow related before we can discover behavioral constraints between activities.

In our discovery approach, the basic idea for relating events comes from the data perspective. As mentioned, each event normally refers to some objects in its corresponding object model, and these reference relations build a bridge between the data perspective and the behavioral perspective. In this section, we refer to the reference relations as AOC relationships which relate (A)civities to (O)bject (C)lasses.

Based on the class model and the AOC relationships, we can relate events to discover the constraints between activities (cf. Section III-C).

**Definition 6 (Discovery of AOC Relationships):** Let \( L = (E, \text{act}, \text{attrE}, EO, om, \subseteq) \) be an XOC log. AOC \( \subseteq U_{OA} \times U_{OC} \) is a set of relationships relating activities to object classes discovered from \( L \) such that AOC = \{\((a, oc) \mid \exists (e, o) \in EO : act(e) = a \land \text{class}(o) = oc\)\}. \( U_{AOC} \) is the universe of AOC relationships.

AOC relationships connect the data perspective with the behavioral perspective. However, they are not powerful enough to describe the cardinality restrictions between events and corresponding objects. For instance, they cannot describe the constraint that one “create order” event should correspond to precisely one “order” object. Therefore, we use cardinality constraints (similar to those on the class relationships) to specify these restrictions to complement AOC relationships.

**Definition 7 (Discovery of AOC Cardinalities):** Let \( L = (E, \text{act}, \text{attrE}, EO, om, \subseteq) \) be an XOC log. AOC \( \subseteq U_{A} \times U_{OC} \) be a set of AOC relationships discovered from \( L \). \( \rho_{AOC} \) and \( \mu_{AOC} \) are discovered cardinality functions from \( L \) where:

- \( \rho_{AOC} \in AOC \rightarrow U_{Card} \) gives the “always” cardinality of the source (activity side) of an AOC relationship such that \( \forall (a, oc) \in AOC : \rho_{AOC}((a, oc)) = \{n \mid \exists e \in E : (\exists o \in o_{oc}(Obj_e) : \exists e' \in E : n = |\{e'' \in o_{oc}(E) : (e'', o) \in EO\}|)\} \).

- \( \mu_{AOC} \in AOC \rightarrow U_{Card} \) gives the “eventually” cardinality of the source (activity side) of an AOC relationship such that \( \forall (a, oc) \in AOC : \mu_{AOC}((a, oc)) = \{n \mid \exists e \in E : (\exists o \in o_{oc}(Obj_e) : n = |\{e' \in o_{oc}(E) : (e', o) \in EO\}|)\} \).

- \( \rho_{OC} \in AOC \rightarrow U_{Card} \) gives the cardinality of the target (object-class side) of an AOC relationship such that \( \forall (a, oc) \in AOC : \rho_{OC}(a, oc) = \{n \mid \exists e \in Ed(E) : n = |\{o \in o_{oc}(Obj_e) : (e, o) \in EO\}|)\} \).

Based on the motivating log, Figure 4 illustrates the discovered AOC relationships and corresponding cardinalities. If an event refers to an object, the activity of the event refers to the class of the object. For instance, since event cs1 refers to three objects ol1, ol2, and ol3, activity create order (i.e., co) refers to class order (i.e., o) and order line (i.e., ol), which means two AOC relationships (create order, order) (see 5) and (create order, order line) (see 6) can be discovered as shown in Figure 4.

For each AOC relationship \((a, oc)\), its target cardinalities can be determined by incorporating numbers of referred oc objects by each a event. Consider \( \mu_{OC}(\subseteq) \) for example. As cs1 has one referred order line object (ol1) and cs2 has two referred order line objects (ol1 and ol2), the directly discovered cardinality is \{1, 2\} and it is extended to \{1, 2, \ldots\} (denoted as \(1^*\)).

In addition, the “always” cardinality on the activity side can be determined by incorporating numbers of happened a events referring to each oc object at every moment after the object is created.\(^8\) Consider \( \rho_{AOC}(\subseteq) \) for example. In the motivating log, there are four order line objects, i.e., ol1, ol2, ol3 and ol4. ol1 is created at the moment cs1 happens and it is not referred to by any create shipment event. Then ol1 is referred to by cs1 at the moment cs1 happens. Afterwards, ol1 is referred to by both cs1 and cs2 at the moment cs2 happens. In summary, ol1 is referred to by (i) zero create shipment event before cs1, (ii) one create shipment event between cs1 and cs2 and (iii) two create shipment events after cs2. Based on the above discussion, the possible cardinality values for ol1 are \{0, 1, 2\}. Using the same method, we incorporate all values for ol1, ol2, ol3 and ol4, and the resulting cardinality is \{0, 1, 2\} which is generalized to \(1^*\).

The “eventually” cardinality on the activity side can be derived by incorporating numbers of all a events (in the whole log) referring to each oc object. Take the AOC relationship \(\subseteq\) as an example. ol3 is referred to by two create shipment events cs1 and cs2 while each of ol2, ol3 and ol4 is referred to by one create shipment event. Hence, the discovered cardinality is \{1, 2\} which is generalized to \(1^{**}\).

**C. DISCOVERY OF BEHAVIORAL MODELS**

In this section, we discover the behavioral model based on the class model and AOC relationships to integrate the OCBC model. More precisely, we discover a set of constraints between activities.

\(^8\)A moment indicates the time when one event just happens. The moments after one object is created include the moment when it first appears and all later moments.
1) RELATING EVENTS

As we mentioned, unlike traditional event logs [30], we do not assume an explicit case notion to relate events. Instead, we use the objects in object models and AOC relationships as the intermediary to relate events. Our goal is to discover a constraint between each two activities, and for that we appoint a reference activity and a target activity to draw a boundary for relating events. Based on this idea, we propose two ways to identify the target events for a reference event.

The first way to relate events is by a triangle pattern. If two events refer to one common object, they are related. For instance, assuming “create order” is the reference activity and “create shipment” is the target activity, we extract all involved events from the example log in Table 1. Based on the “References” column, we draw a graph presenting the reference relations shown in Figure 5(a). As we can see in the “References” column, we draw a graph presenting the involved events from the example log in Table 1. Based on the “References” column, we draw a graph presenting the reference relations shown in Figure 5(a). As we can see in the “References” column, we draw a graph presenting the involved events from the example log in Table 1. Based on the “References” column, we draw a graph presenting the reference relations shown in Figure 5(a). As we can see in the “References” column, we draw a graph presenting the involved events from the example log in Table 1. Based on the “References” column, we draw a graph presenting the reference relations shown in Figure 5(a). As we can see in the “References” column, we draw a graph presenting the involved events from the example log in Table 1. Based on the “References” column, we draw a graph presenting the reference relations shown in Figure 5(a). As we can see in the “References” column, we draw a graph presenting the involved events from the example log in Table 1. Based on the “References” column, we draw a graph presenting the reference relations shown in Figure 5(a). As we can see in the “References” column, we draw a graph presenting the involved events from the example log in Table 1. Based on the “References” column, we draw a graph presenting the reference relations shown in Figure 5(a). As we can see in the “References” column, we draw a graph presenting the involved events from the example log in Table 1. Based on the “References” column, we draw a graph presenting the reference relations shown in Figure 5(a). As we can see in the “References” column, we draw a graph presenting the involved events from the example log in Table 1. Based on the “References” column, we draw a graph presenting the reference relations shown in Figure 5(a). As we can see in the “References” column, we draw a graph presenting the involved events from the example log in Table 1.

![Figure 5](image1.jpg)

**FIGURE 5.** Given a reference event, we navigate to the target events through a triangle pattern.

![Figure 6](image2.jpg)

**FIGURE 6.** Given a reference event, we navigate to the target events through a square pattern.

- triangleP = \{ (a_{ref}, a_{tar}, oc) | \exists oc \in OC : (a_{ref}, oc), (a_{tar}, oc) \subseteq AOC \}
- squareP = \{ (a_{ref}, a_{tar}, rt) | \exists rt \in OC : (a_{ref}, oc), (a_{tar}, oc) \subseteq AOC \wedge \exists rt \in RT : ((a_{ref}, rt), (a_{tar}, rt)) \subseteq AOC \vee ((a_{ref}, \pi_2(rt)), (a_{tar}, \pi_2(rt)) \subseteq AOC) \}

Function extP extracts a set of candidate patterns, in which one candidate pattern is a tuple \((a_{ref}, a_{tar}, ocrt)\) where \(a_{ref}\) is the reference activity, \(a_{tar}\) is the target activity, and \(ocrt\) is a class (corresponding to a triangle pattern) or a class relationship (corresponding to a square pattern) which serves as the intermediary to relate events. Taking the class model and AOC relationships in Figure 4 as input, the function extP returns 6 candidate patterns, i.e., \((create\ order, create\ invoice, order\ line)\), \((create\ order, order\ line, create\ invoice, r8)\), \((create\ invoice, create\ payment, r2)\), \((create\ shipment, create\ order, order\ line)\), \((create\ invoice, create\ order, r8)\), \((create\ payment, create\ invoice, r2)\), in which the first two patterns correspond to the patterns in Figure 5(b) and Figure 6(b) respectively.

**Definition 9 (Event Correlation and Instances):** Let \(L = (E, act, attrE, relate, om, \leq)\) be an XOC log and \(P = (a_{ref}, a_{tar}, cr)\) be a correlation pattern. Function \(extL \in \mathcal{U}_L \times \mathcal{U}_P \rightarrow \mathbb{P}(E^*)\) correlates events in \(L\) based on \(P\) and returns a set of instances (i.e., event sequences), such that \(extL(L, P) = \{ \text{ins} \in E^* \mid (\exists \text{e}_{ref} \in \text{oc}_{ref}(E) : \text{e}_{tar}(\text{ins}) = \{ \text{e}_{ref} \} \cup \mathcal{E}_{L}) \wedge (\forall 1 \leq i < \text{ins} \Rightarrow \text{ins}_i \prec \text{ins}_{i+1}) \} \)

- \(E_{\text{tar}} = \{ \text{e}_{\text{tar}} \in \text{oc}_{\text{tar}}(E) \mid \exists \text{e} \in E, o \in \text{oc}_{cr}(\text{Obj}_{e}) : o \in \text{relate}(\text{e}_{ref}) \cap \text{relate}(\text{e}_{tar}) \} \) if \(P\) is a V pattern, or
- \(E_{\text{tar}} = \{ \text{e}_{\text{tar}} \in \text{oc}_{\text{tar}}(E) \mid \exists \text{e}_{1} \in \text{relate}(\text{e}_{ref}), \text{e}_{2} \in \text{relate}(\text{e}_{tar}), e \in E : (\{cr, o_1, o_2\}, cr, o_1, o_2) \cap \text{rel}_e = \emptyset \}\) if \(P\) is a U pattern.

For simplicity, we define \(\text{ins}_{P} = (\text{before}, \text{after}) = (\text{\langle e}_{e_{ref}(E_{\text{tar}})}, \text{\rangle e}_{e_{ref}(E_{\text{tar}})})\).

Each candidate pattern specifies a way to correlate events. Based on the extracted patterns, we correlate events in the motivating log, resulting in a graph in Figure 7. More precisely, the 10 events are sorted by time increasingly from left to right (e.g., \(co1\) is before \(co2\)). The object(s) in the parentheses attached on edges between events indicate how the events are correlated. For instance, the four highlighted edges correspond to the examples in Figure 5(a) and Figure 6(a).
2) BEHAVIORAL CONSTRAINTS

As mentioned, the result using the relation method is a set of target events for each reference event, rather than a process instance (e.g., a case) which includes events over the whole process. Therefore, we cannot use strict procedural constraints such as those in Petri nets to form the behavioral model. Instead, we employ a declarative way to indicate the temporal restriction between a reference event and its target events. A temporal restriction may cover two perspectives, i.e., it may restrict the number of target events both before and after the reference event. In this paper, we will employ a graphical notation, i.e., a set of constraint types inspired by Declare (a declarative workflow language [33]), to represent and visualize the restrictions.

Definition 10 (Constraint Types): \( \mathcal{UCT} = \{X \subseteq \mathbb{N} \times \mathbb{N} | X \neq \emptyset\} \) defines the universe of all possible constraint types. Any element of \( \mathcal{UCT} \) is a constraint type which specifies a non-empty set of pairs of integers: the first integer defines the number of target events before the reference event and the second integer defines the number of target events after the reference event. Figure 8 shows the graphical notations of 8 example constraint types.

A constraint type assigns a restriction on the target events. In our approach, if the observed target events follow the restriction corresponding to some constraint type, this type is considered as a discovered constraint type.

Definition 11 (Specification of Behavioral Constraints): Let \( P = (a_{ref}, a_{tar}, cr) \) be a correlation pattern and \( ct \) be a constraint type. A behavioral constraint is specified as a tuple \( (P, ct) \in U_P \times U_CT \). Function \( idCon \in U_P \times U_CT \rightarrow U_{Con} \) gives a constraint id referring to a constraint specification. Furthermore, function \( speCon \in U_{Con} \rightarrow U_P \times U_CT \) gives the constraint specification corresponding to a constraint id. By relating events based on a correlation pattern, we get a set of instances, which can be used to discover constraints corresponding to the pattern. The semantics of a constraint is defined as a restriction on the relations between events in a scope. On the other hand, from the discovery angle, a constraint can be identified as a tuple of a pattern and a constraint type, i.e., \( (P, ct) \). The pattern \( P \) specifies the scope, i.e., the reference events and target events, and the constraint type \( ct \) indicates the restriction on the relations between reference events and target events.

Definition 12 (Discovery of Behavioral Constraints): Let \( L = (E, act, attrE, relate, om, \geq) \) be an XOC log, \( P = (a_{ref}, a_{tar}, cr) \) be a correlation pattern and \( CT \) be a set of constraint types. Function \( disCon : U_L \times U_P \times \mathcal{P}(CT) \rightarrow \mathcal{P}(U_{Con}) \) discovers a set of behavioral constraints, such that \( disCon(L, P, CT) = \{c | c = idCon(P, ct) \land ct \in CT \land \forall e \in extI(L, P) : ins(P, ct) \in ct \} \).

3) BEHAVIORAL CONSTRAINT MODELS

After illustrating the method to relate events in Section III-C1 and the notation to describe behavioral constraints in Section III-C2, we can now define and discover behavioral models based on them.

Definition 13 (Discovery of Behavioral Constraint Model): Let \( L \) be an XOC log, \( CT \) be a set of constraint types, \( ClaM \) be a discovered class model and \( AOC \) be a discovered set of AOC relationships. A behavioral constraint model discovered from \( L \) is a tuple \( BCM = (A, C, \pi_{ref}, \pi_{tar}, type) \), where

- \( A \subseteq \mathcal{U}_A \) is the set of activities such that \( A = \{act(e) | e \in E\} \),
- \( C \subseteq \mathcal{U}_{Con} \) is a set of constraints such that \( C = \{c | c \in disCon(L, P, CT) \land P \in extP(ClaM, AOC)\} \),
- \( \pi_{ref} \in C \rightarrow A \) defines the reference activity of a constraint such that \( \forall c \in C : \pi_{ref}(c) = a_{ref}, \) where \( ((a_{ref}, a_{tar}, cr), ct) = speCon(c) \),
- \( \pi_{tar} \in C \rightarrow A \) defines the target activity of a constraint such that \( \forall c \in C : \pi_{tar}(c) = a_{tar}, \) where \( ((a_{ref}, a_{tar}, cr), ct) = speCon(c) \), and
- \( type \in C \rightarrow UC_{T} \) defines the type of a constraint such that \( \forall c \in C : type(c) = ct, \) where \( ((a_{ref}, a_{tar}, cr), ct) = speCon(c) \).

A behavioral constraint model is a collection of activities and constraints. A constraint corresponds to a constraint type, a reference activity (on the side with a dot) and a target activity (on the other side), which indicates a restriction on both the numbers of target events before and after each reference event.

For instance, Figure 9(a) shows a behavioral model example, i.e., \( BCM = (A, C, \pi_{ref}, \pi_{tar}, type) \) with \( A = \{create order, create invoice\}, C = \{c_1, c_2\}, \pi_{ref}(c_1) = create order, \pi_{ref}(c_2) = create invoice, \pi_{tar}(c_1) = create invoice, \pi_{tar}(c_2) = create order, type(c_1) = response, and type(c_2) = precedence. c_1 \) indicates that each “create order” event should have at least one related “create invoice” event after it and \( c_2 \) indicates that each “create invoice” event should have at least one related “create order” event before it.

Note that it is possible to merge two constraints with the same shape (e.g., two arrows) and opposite reference activity and target activity into one constraint for simplicity.

---

**FIGURE 7.** The extracted events from the motivating example.

**FIGURE 8.** Graphical notation for 8 example constraint types. The dot on the left-hand side of each constraint refers to the reference events. Target events are on the other side that has no dot. The notation is inspired by Declare, but formalized in terms of cardinality constraints rather than LTL.
instance, c12 in Figure 9(b) is a combined constraint, which is equal to the two constraints c1 and c2 in Figure 9(a).

D. OBJECT-CENTRIC BEHAVIORAL CONSTRAINT MODEL

Up to now, we have explained all the elements of an OCBC model and corresponding discovery approach. More precisely, Section III-A focused on structuring objects and discovering cardinality constraints on class models while Section III-B built a bridge for discovering behavioral constraints based on class models. Section III-C related events by elements in object models and discovered behavioral constraints between each pair of activities identified by the candidate patterns. Here, we give the formal definition of an OCBC model and discuss it from different perspectives based on a real scenario.

Definition 14 (Discovery of OCBC Model): Let \( L = (E, act, attrE, EO, om, \leq) \) be an OOC log. An object-centric behavioral constraint model discovered from \( L \) is a tuple \( OCBCM = (Clam, AOC, \pi_A, \pi_O, \pi_C, BCM, crel) \), where

- \( Clam \) = \( (OC, RT, \pi_1, \pi_2, \pi_{ref}, \pi_{tar}, type) \) is the class model discovered from \( L \) (Definition 5),
- \( AOC \subseteq A \times OC \) is the set of AOC relationships discovered from \( L \) (Definition 6),
- \( \pi_A, \pi_O \in AOC \rightarrow \mathcal{U}_{Card} \) are three cardinality functions discovered from \( L \) (Definition 7),
- \( BCM = (A, C_{ref}, C_{tar}, type) \) is a behavioral constraint model discovered from \( L \) (Definition 13), and
- \( crel \in C \rightarrow OC \cup RT \) is the constraint relation such that

\[ \forall c = (a_1, a_2, ocr, ct) \in C : crel(c) = ocr. \]

\( \mathcal{U}_{OCBCM} \) is the universe of OCBC models.

Figure 10 shows a discovered OCBC model from the motivating log, which describes the data perspective, behavioral perspective and the interplay between them in a single diagram. In general, it clearly reveals the involved classes, activities and constraints in the OTC scenario from which the motivating log is generated.

The 9 classes and 10 class relationships make up the backbone of the OCBC model. They indicate the involved objects and restrictions which are followed in the real transactions. For instance, \( r^9 \) indicates that each “order line” is eventually packed in some shipment lines (indicated by \( \diamond 1..* \)), which is consistent with the rule in the OTC scenario.

The 8 AOC relationships (i.e., \( 1, ..., 8 \)) indicate how the behavioral perspective (i.e., activities) refers to the data perspective (i.e., classes). Besides, they describe the restriction between events and objects in the OTC scenario through cardinality constraints. For instance, \( \circ \) indicates a one-to-one correspondence between order objects and create order events through the constraint \( \square 1, \diamond 1 \) and 1, which means that if an object is added to the class order, the corresponding activity is also executed and vice versa. \( \square^* \) of \( \circ \) shows that, at any moment, each order line may have an arbitrary number of “create shipment” events while \( \diamond 1..* \) of \( \circ \) means that each order line should have at least one “create shipment” event eventually. In addition, the constraint on the class side of \( \circ \) (\( 1..* \)) denotes that one “create shipment” event may send products from one or more order lines.

The discovered model also contains 8 behavioral constraints (i.e., c1..c8) which specify the temporal restrictions on behavioral perspective. For example, c7 indicates that each “create order” event should have at least one related “create shipment” event after it and c8 indicates that each “create shipment” event should have precisely one related “create order” event before it. Each constraint is identified by a candidate pattern and we use constraint relations (denoted by a dotted line between a behavioral constraint and a class or a class relationship) to indicate the class or relationship (i.e., the intermediary to relate events) of the pattern. For instance, there is a constraint relation between c1 and r2, which means c1 is identified by the pattern (create payment, create invoice, r2).

IV. ADVANCED DISCOVERY TECHNIQUES TO DEAL WITH NOISE

Model discovery techniques take an event log as input and return a model to reveal the real business process where the log is generated. The quality of the input log decides if the discovered model can really represent the real business process. In other words, to discover a representative model, the event log should contain a representative sample of data. A sample may be not suitable due to problems from two perspectives: (i) it has “too little data” to cover various events and objects in the process; (ii) it has “too much data” with events or objects unrelated to the target process. In process mining, the first perspective refers to the “incompleteness” problem while the second perspective refers to the “noise” problem.

In this section, we focus on dealing with the “noise” problem, which is significant in process mining and should be faced by any applicable discovery algorithm. Note that the noise here refers to the infrequent events or objects rather than the incorrect logging, since we assume all the data in logs are correctly recorded.

The approach in Section III cannot deal with noise and it mainly suffers two problems when facing noise. First, the noise may contain a variety of infrequent events or objects
unrelated to the target business process, which makes the discovered models too complex (i.e., too many entities and constraints). Besides, the discovered constraints are not precise, since the approach forces the constraints to allow the occurrence of noise. In order to solve these problems, we next introduce some more advanced approaches to deal with noise in logs by (i) simplifying complex discovered models, (ii) discovering precise behavioral constraints and (ii) discovering precise cardinality constraints.

A. SIMPLIFICATION OF OCBC MODELS

The approach in Section III is sensitive to noise, e.g., one object of an infrequent class leads to the discovery of the “noise” class. As a result, the noise makes the useful insights hiding in the discovered complex model. In this section, we give solutions to simplify OCBC models, i.e., filtering the unnecessary entities and constraints corresponding to the infrequent events or objects, to make the insights outstanding.

Definition 15 (Support): Let \( L = (E, act, attrE, relate, con, \leq) \) be an XOC log, \( M = (\text{ClaM}, \text{ActM}, \text{AOC}, \pi_A, \pi^{\text{rel}}, \pi^{\text{OC}}, \pi^{\text{rel}, \pi^{\text{OC}}, \pi^{\text{act}, \pi^{\text{OC}}, \pi^{\text{rel}, \pi^{\text{OC}}, \pi^{\text{act}, \pi^{\text{OC}}, \pi^{\text{rel}, \pi^{\text{OC}}, \pi^{\text{act}}}}}}}} \) be the discovered OCBC model where \( \text{ClaM} = (C, R, \pi_1, \pi_2, \pi^{\text{rel}}, \pi^{\text{act}, \pi^{\text{rel}}, \pi^{\text{act}, \pi^{\text{rel}, \pi^{\text{act}, \pi^{\text{rel}, \pi^{\text{act}}}}}}}} \) is the class model and \( \text{ActM} = (A, Con, \pi^{\text{ref}, \pi^{\text{tar}}, \text{type}} \) is the activity model. \( \text{Ele}_M = C \cup R \cup A \cup \text{AOC} \cup Con \) is the set of all elements in the model. Function \( \text{support} \in \text{Ele}_M \rightarrow [0, 1] \) maps an element \( ele \in \text{Ele}_M \) to a value between 0 and 1. The function has five versions to calculate the support of different types of elements.

The support of a class is defined as the fraction of objects of the class in all objects, i.e., for each \( c \in C \),

\[
\text{supportC}(c) = \frac{|\{o | \exists e \in E : (o \in \text{Obj}_e \land \text{class}_e(o) = c)\}|}{|\{o | \exists e \in E : o \in \text{Obj}_e\}|}.
\]

The support of a class relationship is defined as the fraction of object relations corresponding to the relationship in all object relations, i.e., for each \( r \in R \),

\[
\text{supportR}(r) = \frac{|\{(r, o_1, o_2) | \exists e \in E : (r, o_1, o_2) \in \text{Rel}_e\}|}{|\text{rel} | \exists e \in E : \text{rel} \in \text{Rel}_e|}.
\]

The support of an activity is defined as the fraction of events of the activity in all events, i.e., for each \( a \in A \),

\[
\text{supportA}(a) = \frac{|\{e \in E | \text{act}(e) = a\}|}{|E|}.
\]

The support of an AOC relationship is defined as the fraction of references corresponding to the relationship in all references, i.e., for each \( aoc = (a, c) \in \text{AOC} \), as shown at the bottom of the page.

The support of a behavioral constraint is defined as the fraction of instances of \( (P) \) which consist with the constraint,

\[
\text{supportAOC}(aoc) = \frac{|\{(e, o) | e \in E \land o \in \text{relate}(e) \land \text{act}(e) = a \land \text{class}_e(o) = c\}|}{|(e, o) | e \in E \land o \in \text{relate}(e)|}.
\]

Consider the log in Table 1 and the model in Figure 10 to understand how to compute the support of different elements. For instance, \( \text{supportC(order)} = 2/28 \) since there are two \( \text{order} \) objects and twenty-eight objects in total. \( \text{supportR} (r10) = 4/36 \) since there are four object relations of \( r10 \) and thirty-six object relations in total. \( \text{supportA(create order)} = 2/10 \) since there are two \( \text{create order} \) events and ten events in total. \( \text{supportAOC(aoc5)} = 2/26 \) since there are two references of \( aoc5 \) (between \( \text{create order} \) events and \( \text{order} \) objects) and twenty-six references in total. \( \text{supportAOC(aoc2)} = 3/15 \) since there are three consistent instances of \( P2 \) (corresponding pattern of \( \text{con2} \)) and fifteen instances in total.

Note that the support in Definition 15 is a relative support, i.e., a fraction between 0 and 1. It is possible to compute an absolute support by only considering the numerator of each formula, e.g., \( \text{supportC}(c) = |\{o | \exists e \in E : (o \in \text{Obj}_e \land \text{class}_e(o) = c)\}| \) for a class \( c \). Table 2 presents the support of all classes in Figure 10, i.e., the “Instance number” column shows the absolute support and the “Support” column shows the relative support. One can specify a threshold for support to filter out any element whose support is below the threshold. In this way, we can simplify the complex OCBC model discovered in the environment with noise.

Definition 16 (Filtering Infrequent Elements in OCBC Model): Let \( L \) be an XOC log, \( M \) be the discovered OCBC model and \( \text{Ele}_M = C \cup R \cup A \cup \text{AOC} \cup Con \) be the set of all elements in the model. Let \( \tau \) be a threshold. An element in the model \( ele \in \text{Ele}_M \) is filtered out, if its corresponding support is below the threshold, i.e., \( \text{support}(ele) < \tau \) and all its related elements are removed too as follows:

- if \( \text{ele} = c_{del} \in C \), the related elements are \( \text{R}_{del} = \{r \in R | \pi_1(r) = c \lor \pi_2(r) = c\} \) and \( \text{AOC}_{del} = \{a, c_{del} \in \text{AOC}\} \) and \( \text{Con}_{del} = \{\text{con} \in \text{Con} \mid \text{crel}(\text{con}) \subset c_{del} \lor \text{crel}(\text{con}) \subset R_{del}\} \).
- if \( \text{ele} = r_{del} \in R \), the related elements are \( \text{Con}_{del} = \{\text{con} \in \text{Con} \mid \text{crel}(\text{con}) = r_{del}\} \).
- if \( \text{ele} = a_{del} \in A \), the related elements are \( \text{AOC}_{del} = \{(a_{del}, c) \in \text{AOC}\} \) and \( \text{Con}_{del} = \{\text{con} \in \text{Con} \mid \pi^{\text{ref}}(\text{con}) = a_{del} \lor \pi^{\text{tar}}(\text{con}) = a_{del}\} \).
TABLE 2. Support and related elements of classes.

| Class     | Instance number | Support | $R_{del}$ | $AOC_{del}$ | $Con_{del}$ |
|-----------|-----------------|---------|-----------|-------------|-------------|
| payment   | 2               | 2/28    | r7        | aoc2        | -           |
| payment line | 3             | 3/28    | r7,r2     | aoc1        | con1,con2,con3 |
| element relation | 4       | 4/28    | r3,r8     | aoc3        | con4,con5   |
| invoice   | 3               | 3/28    | r1,r3     | aoc4        | con1,con2,con3 |
| order     | 2               | 2/28    | r8,r10    | aoc5        | con4,con5   |
| customer  | 2               | 2/28    | r1,r4,r5  | -           | -           |
| order line | 4               | 4/28    | r9,r10    | aoc6,aoc7   | con6,con7,con8 |
| shipment line | 5            | 5/28    | r6,r9     | -           | -           |
| shipment  | 3               | 3/28    | r4,r6     | aoc8        | -           |

- if $ele = (a, c)_{del} \in AOC$, the related elements are $Con_{del} = \{con \in Con \mid (\pi_{ref}(con) = a \lor \pi_{tar}(con) = a) \land (crel(con) = c \lor \pi_1(crel(con)) = c \lor \pi_2(crel(con)) = c)\}$, and
- if $ele = con_{del} \in Con$, there are no related elements.

Note that in the process of discovering an OCBC model, the elements are discovered in the order: $C$, $R$, $A$, $AOC$ and $Con$. As a result, the latter discovery may depend on the former discovery result. For instance, the discovery of class relationships depends on the discovered classes. Therefore, when an element $ele$ is filtered out, all its related elements (i.e., elements discovered based on $ele$) should be removed too. Based on the type of $ele$. Definition 16 gives five rules to specify the related elements.

If $ele$ is a class $c_{del}$, the related elements consist of (i) all class relationships connected to the class ($R_{del}$), (ii) all AOC relationships connected to the class ($AOC_{del}$), and (iii) all behavioral constraints which correlate events by $c_{del}$ or a class relationship which will be removed ($Con_{del}$). If $ele$ is a class relationship $r_{del}$, the related elements only consist of behavioral constraints which correlate events by $r_{del}$. If $ele$ is an activity $a_{del}$, the related elements contain AOC relationships which are related to the activity ($AOC_{del} = \{(a_{del}, c) \in AOC\}$) and behavioral constraints which have the activity as reference or target activity ($\pi_{ref}(con) = a \lor \pi_{tar}(con) = a$). If $ele$ is an AOC relationship $(a_c, c_{del})$, any behavioral constraint $con$ which is connected to $a$ (i.e., $\pi_{ref}(con) = a \lor \pi_{tar}(con) = a$), and correlate events by $c$ (i.e., $crel(con) = c$) or a class relationship connected to $c$ (i.e., $\pi_1(crel(con)) = c \lor \pi_2(crel(con)) = c$) will be removed too. Since the constraints are the last elements to be discovered, removing a constraint will not influence anything else.

For instance, for the class $payment$ in Figure 10, its related elements are $r7$ and $aoc2$ since they are connected to $payment$. It has no related behavioral constraints since it is not involved in any correlation pattern. Table 2 presents the related elements of all classes in Figure 10. Based on the support and related elements in Table 2, one can set a threshold to filter classes. For instance, given a threshold 3.5/28, the classes $payment$, $payment line$, $invoice$, $order$, $customer$ and $shipment$, and their related elements are removed, resulting in the model in Figure 11. Note that it is also possible to set a different threshold for each type of elements. The rules in Definition 16 still hold in these situations.

B. DISCOVERY OF PRECISE BEHAVIORAL CONSTRAINTS

In Section IV-A, we proposed an approach to simplify the complex models, which deals with the noise problem on the simplicity perspective. Besides the simplicity perspective, the noise may force the constraints to be general enough to allow its occurrence, resulting in imprecise constraints. In this section, we illustrate how to filter noise and discover precise behavioral constraints, i.e., dealing with the noise problem on the imprecision perspective.

A constraint is defined in the context of a correlation pattern. It indicates the restriction on the reference events and target events in instances correlated by the pattern. In the discovery process, the observed instances decide the discovered constraints, i.e., if all instances (corresponding to the correlation pattern) satisfy the semantics of some constraint type, a constraint of this type is discovered. The requirement that all instances satisfy the semantics is too strict in an environment with noise, since an infrequent behavior, which is only a violation of a constraint type, may ruin the discovery of the corresponding constraint. Therefore, it is necessary to propose a robust discovery approach of behavioral constraints, which can still discover a constraint event if some noise violates the constraint.

The basic idea is to filter the noise according to a threshold, and discover constraints based on the behavior without noise as shown in Figure 13. In this way, the discovered constraints are precise, i.e., they do not allow the occurrence of noise. In this paper, the noise in terms of behavioral constraints refers to infrequent instances. We define the types (i.e., variants) of instances and count the frequency of each type to identify the noise. If the frequency of a type is below the configured threshold, the instances of the type are considered as noise and filtered out. The remaining frequent types (highlighted in black) are taken as input to discover behavioral constraints. Next, we define a variant matrix to represent all possible types of instances and count the frequency of each type.
A variant matrix is represented by a grid consisting of 3 x 3 cells.

Definition 17 (Variant Matrix): A variant matrix $V_{CT}$ is a set of nine disjoint constraint types which incorporate all the possible relations between a reference event and its target events. $V_{CT} = \{(0; 0), (1; 0), (0; 1), (1; 1), (2+; 0), (0; 2+), (2+; 1), (1; 2+), (2+; 2+)\} \subseteq U_{CT}$. Each constraint type in the set is called a variant, e.g., $(1; 1) \in V_{CT}$ is a variant.

The variant matrix contains nine different variants (cells), in the form of three rows and three columns as shown in Figure 12. Each variant in the matrix corresponds to a constraint type. For instance, $(1; 0)$ is a constraint type which requires that the number of target events before (after) each reference event is one (zero). $(1; 1)$ is the same as the unprecedence constraint type. $(2+; 0)$ is a constraint type which requires that the number of target events before each reference event is larger or equal to two, and there are no target events after each reference event. The formalization of these nine constraint types is shown in Table 3.

Note that the nine variants are disjoint, i.e., $\forall v, v' \in V_{CT}: v \neq v' \Rightarrow v \cap v' = \emptyset$. Besides, the nine variants cover all the possible relations between a reference event and its target events, i.e., each instance corresponds to precisely one variant, indicated by the relations between target events and the reference event in the instance. For example, if an instance has five target events before the reference event and precisely one target event after the reference event, i.e., $(5, 1)$, it corresponds to variant $(2+; 1)$ (since $(5, 1) \in (2+; 1)$). Note that $(5, 1)$ is a pair of numbers (in which the former (latter) one represents the number of target events before (after) the reference event) while $(2+; 1)$ is a set of pairs.

Definition 18 (Computing Variant Frequency): Let $L$ be an XOC log and $P$ be a correlation pattern. Function $\text{freV} \in \mathcal{U}_L \times \mathcal{M}_P \times V_{CT} \rightarrow \mathbb{N}$ returns the frequency that a variant is observed in a log corresponding to a pattern such that $\text{freV}(L, P, v) = |\{\text{ins} \mid \text{ins} \in \text{extI}(L, P) \land \text{ins} \in v \in P\}|$.

For convenience, we define the following shorthand. $\text{freV}_{\text{rel}}(L, P, v) = \frac{\text{freV}(L, P, v)}{\sum_{v' \subseteq V_{CT}} \text{freV}(L, P, v')}$. Provides the ratio of a variant.

If an instance correlated by a correlation pattern $P$ from a log $L$ corresponds to a variant $v$, i.e., ins$_P \in v$, we say that $v$ is observed once in $L$. Based on this idea, Definition 18 defines a function to count how many times a variant is observed in a log corresponding to a pattern. The frequency of a variant is equal to the number of instances which satisfy the requirements of the variant.

For instance, let $L$ be the log example in Table 1 and $P = (\text{create order}, \text{create shipment}, \text{order line})$. After event correlation, we get two instances $\{(\text{co1}, \text{cs1}, \text{cs2}), (\text{co2}, \text{cs3})\}$. The first solution to identify frequent variants is based on the frequency of individual variants and the second solution also considers the total frequency (i.e., a relative ratio).

Table 3. The formalization of nine variants in the variant matrix.

| constraint variant | formalization |
|-------------------|---------------|
| $(0; 0)$ | $(\text{before, after})$ (before = 0 ∧ after = 0) |
| $(1; 0)$ | $(\text{before, after})$ (before = 1 ∧ after = 0) |
| $(2+; 0)$ | $(\text{before, after})$ (before > 1 ∧ after = 0) |
| $(0; 1)$ | $(\text{before, after})$ (before = 0 ∧ after = 1) |
| $(1; 1)$ | $(\text{before, after})$ (before = 1 ∧ after = 1) |
| $(2+; 1)$ | $(\text{before, after})$ (before > 1 ∧ after = 1) |
| $(0; 2+)$ | $(\text{before, after})$ (before = 0 ∧ after > 1) |
| $(1; 2+)$ | $(\text{before, after})$ (before = 1 ∧ after > 1) |
| $(2+; 2+)$ | $(\text{before, after})$ (before > 1 ∧ after > 1) |

Definition 19 (Frequent Variants): Let $L$ be an XOC log and $P$ be a correlation pattern. $V_{\text{fre}} \subseteq V_{CT}$ is the frequent variants observed in $L$ corresponding to $P$, such that for any $v \in V_{\text{fre}}$

- $\text{freV}(L, P, v) \geq \tau$ for some threshold $\tau \in \mathbb{N}$, or
- $\text{freV}_{\text{rel}}(L, P, v) \geq \tau$ for some threshold $\tau \in [0, 1]$, or
- $\text{freV}_{\text{Entropy}}(L, P, v) \geq \tau$ for some threshold $\tau \in [0, 1]$, where $\text{freV}_{\text{Entropy}}(L, P, v) = P \sum_{v' \subseteq V_{CT}} \text{log}(p' \text{log}(p')) - P \sum_{v' \subseteq V_{CT}} \text{log}(\text{freV}_{\text{rel}}(L, P, v'))$.

Definition 19 provides three solutions to identify frequent variants by filtering infrequent variants (i.e., noise) according to a threshold. In the first solution, a variant is frequent if its frequency is equal to or larger than the given threshold (i.e., an integer). Figure 15(a) shows an example using the first solution to identify frequent variants. With a threshold 25, the variants $(1; 1)$ and $(2+; 1)$ are frequent (highlighted in black) since their frequencies (40 and 35) are above the threshold. The second solution employs a similar idea, only changing the absolute frequency to a relative ratio.

The first solution is based on the frequency of an individual variant and the second solution also considers the total frequency (i.e., a fraction of individual frequency in total frequency). They have limitations in some cases, where the frequencies of other variants should be considered when identifying if a variant is frequent. For instance, the variant $(0; 1)$ has the same frequency (i.e., 8) in the frequency matrices in Figure 15(a) and Figure 15(b), and the frequency sums of all variants in these two matrices are also the same (i.e., 100). To identify if variant $(0; 1)$ is frequent in these two different
Definition 20 (Discovery of Behavioral Constraints by Frequent Variants): Let \( L \) be an XOC log, \( P \) be a correlation pattern and \( CT \) be a set of constraint types. Function \( \text{disCN} \in \mathcal{U}_L \times \mathcal{U}_P \times \mathcal{P}(CT) \rightarrow \mathcal{P}(\mathcal{U}_{\text{Con}}) \) discovers behavioral constraints from a log with noise, such that

\[
\text{disCN}(L, P, CT) = \{ \text{con} | \text{con} = \text{idCon}(P, ct) \land ct \in CT_{\text{fre}} \setminus CT_{\text{red}} \}
\]

where

- \( CT_{\text{fre}} = \{ ct \in CT | \forall v \in V_{\text{fre}} : v \subseteq ct \} \), where \( V_{\text{fre}} \subseteq V_{CT} \) is the frequent variants observed in the log, and
- \( CT_{\text{red}} = \{ ct \in CT_{\text{fre}} | \exists ct' \in CT_{\text{fre}} : ct' \subset ct \} \), which is the set of redundant constraint types.

If all frequent variants consist with a constraint type \( ct \) (i.e., \( \forall v \in V_{\text{fre}} : v \subseteq ct \)), a constraint of this type is discovered, i.e., \( \text{con} = \text{idCon}(P, ct) \). Note that it is possible that there exist overlapping constraint types. For instance, the response type contains the unary-response type, i.e., the response type is looser. If two types are contained by \( CT_{\text{fre}} \) and one contains the other, the looser type is called a redundant constraint type. When discovering constraints, redundant types are discarded.

V. IMPLEMENTATION

Our discovery algorithm has been implemented in the “OCBC Model Discovery” Plugin in ProM.\(^{10}\) Figure 16 shows the interface of the plugin and a discovered model (in panel 4) from the example XOC log.\(^{11}\) Note that the discovered model is same as the model shown in Figure 10 except the combined constraints are displayed separately.

Panel 1 presents the distribution of cardinalities and the instances related to one selected constraint (highlighted in red in panel 4). Panel 2 shows the attributes of a selected edge or node. It is possible to zoom in/out models through operating panel 3. Using the filter panel 6, it is possible to filter behavioral constraints based on constraint types (the plugin discovers constraints of 8 types shown in Figure 8) and activity names, to get a simplified model. Panel 5 enables users to guide the model discovery approach, i.e., update (or rediscover) behavioral constrains by selecting approaches of computing metrics and configuring weights (between 0 and 1) for three metrics.

The time of model discovery increases along with the number of events in logs. Figure 17 shows the time for different numbers of events. For example, our plugin takes around 35 seconds to discover a model from a log with 240,000 events. According to the line indicated by the dots, we claim that the time is linear with the numbers of events.

VI. COMPARISON OF OCBC MODELS AND OTHER MODELS

Based on the motivating example, we compare OCBC models with other models such as Declare models, directly follow graphs and Petri nets, and show advantages of our modeling language. The basic idea is to transform the motivating data into XOC logs and XES logs. Then we discover OCBC models from XOC logs and discover other models from XES

\(^{10}\)Access http://www.promtools.org/prom6/nightly/ to download ProM 6 Nightly builds and update the OCBC package.

\(^{11}\)The involved logs are at http://www.win.tue.nl/ocbc/.
logs. The models are compared by checking which models can best describe the business process from where the data comes.

**A. BUSINESS PROCESS**

First, we introduce the business process of the motivating example. Figure 18 employs an informal notation to describe the business process. In general, the arrows of edges indicate the temporal order between activities while the cardinalities on edges indicate the constraints on the instances of activities. For instance, the edge between the “create invoice” activity and the “create payment” activity means that each “create payment” event must occur after its corresponding “create invoice” event, while the cardinalities on the edge means that each “create invoice” event corresponds to precisely one “create payment” event and each “create payment” event corresponds to one or more “create invoice” events.

According to this business process, we operate Dolibarr to generate transactions in the database. By incorporating different numbers of orders, we can extract logs of different sizes as shown in Figure 17. To facilitate understanding, we only show the comparison result based on the data (e.g., two orders) as shown in Figure 1. Based on the data, it is possible to create the XOC log shown in Table 1. Additionally, we can also transform the data into an XES log as shown in Figure 19. We consider the order id as the case id and use the method in [30] to relate events, resulting in two cases o1 and o2. Note that this generated log has convergence problems, e.g., ci2 is contained by two cases as if it is executed twice though it is performed only once in Dolibarr, and divergences problems, e.g., in case o2 “create payment” has two instances cp1 and cp2 which cannot be distinguished in the case though they are performed on different documents in the Dolibarr (i.e., cp1 is on ci2 and cp2 is on ci3) [26], [29].

**B. COMPARISON OF DISCOVERED MODELS**

Based on the generated XOC and XES logs, we compare the data Petri net, BPMN model, Declare model, and directly follow graph (DFG) discovered from the XES log and the OCBC model discovered from the XOC log.

Figure 20 displays a discovered data Petri net based on the approach in [7], [15], and [16]. On the behavioral perspective, the discovered Petri net has a lot of implicit transitions and loops. It is not precise and allows behavior which is not possible in the real business process. For instance, the “create shipment” activity can be skipped in the discovered Petri net, which violates the real process in which each “create order” should be followed by at least one “create shipment” event. Besides, the many-to-many relationship between “create order” and “create invoice” in the process is transformed into a one-to-many relationship. On the data perspective, some variables and guards are discovered, which present the conditions to trigger transitions. The dotted line shows the interactions between activities and variables.

Figure 21 presents a discovered BPMN model using the approach in [6]. The model is not precise, e.g., the sub-process is a “flower” model involving activities “create payment” and “create invoice”, which can happen at any order for any number of times. The approach in [6] only discovers the behavioral perspective with hierarchies, and we do not find any existing work which can automatically discover the data perspective. However, it is possible to manually add (or discover them automatically in the future) variables, data objects and data stores to describe the data perspective.

A discovered data-aware Declare model [20], [22] is shown in Figure 22(a). For simplicity, we limit the discovered constraint types to the set of “response”, “non-response”, “precedence” and “non-precedence”. Declare models use a declarative way to express the constraints between pairs of activities [21]. For instance, the constraint c1 between...
“create invoice” and “create payment” indicates that each “create invoice” event should be followed by a “create payment” event and each “create payment” event should be preceded by a “create invoice” event. The real business process indicates “create payment” events cannot occur before corresponding “create invoice” events. However, a negative constraint (c2) needed to describe this rule is not discovered, since in case o2, the “create payment” event cp1 is wrongly related to its subsequent “create invoice” event ci3 (the divergence problem, cf. Section VI-A). Besides, existing constraint types in Declare models do not have a constraint
type like $c^3$ to restrict the cardinality between "create invoice" and "create payment" events, which is required in the business process. The discovered data perspective consists of guards on constraints, which specify the situations when constraints are enabled. For instance, if we add an instance $o^3 = \langle co^3, ci^4, cp^3 \rangle$, where the attribute $customer$ of $co^3$ is $c^2$, into the log shown in Figure 19, a guard $customer = c^1$ is discovered on the "response" constraint between "create order" and "create shipment", since the "create order" events are followed by "create shipment" events only in $o^1$ and $o^2$ where $customer = c^1$.

A DFG shows the directly-follow relations between activities and possibly also shows the corresponding frequencies. Indicated by Figure 22(b), "create invoice" activity happens 4 times which is not consistent with reality (the convergence problem, cf. Section VI-A). Moreover, the DFG shows that a "create payment" event is directly followed by a "create invoice" event once, which also violates the rule in the real business process.

In comparison, our OCBC model shown in Figure 10 is better able to describe the real business process. On the behavioral perspective, since the XOC log has no convergence and divergence problems (multiple instances can be distinguished when we relate events, cf. Section III-C1), all constraints are correctly discovered. Besides, with the support of a class model and AOC relations, the cardinality constraints (e.g., one-to-many and many-to-many relationships) between activities can also be clearly described. The data perspective is powerfully described with a class model.

VII. CONCLUSION

In this paper, we proposed a discovery approach for a novel model, named Object-Centric Behavioral Constraint (OCBC). OCBC models graphically describe control-flow and data/objects with cardinality constraints in a truly integrated manner. In this way, we overcome the problems of existing approaches that separate behavioral (e.g., BPMN, EPCs, or Petri nets) and the data (e.g., a class model) perspectives. Moreover, OCBC models allow different types of instances to interact in a fine-grained manner and the constraints in the class model guide behavior.

Besides, some metrics such as fitness, precision and generalization are proposed to evaluate discovered models, and users can guide the model discovery algorithm by configuring metric weights. At last, we compare OCBC models with other models such as BPMN, Declare models, DFG and Petri nets, and show that our model can better represent object-centric business processes with one-to-many and many-to-many relations.

Moreover, this paper serves as a starting point for a new line of research. Next to model discovery and its support tools (OCBC Model Editor and OCBC Model Discovery Plugin) in ProM, we also support conformance checking. Based on OCBC models, many deviations which cannot be detected by existing approaches can be revealed.

REFERENCES

[1] R. Addo-Tenkorang and P. Helo, “Enterprise resource planning (ERP): A review literature report,” in Proc. World Congr. Eng. Comput. Sci., vol. 2, 2011, pp. 19–21.
[2] A. Augusto, R. Conforti, M. Dumas, and M. La Rosa, “Split Miner: Discovering accurate and simple business process models from event logs,” in Proc. IEEE Int. Conf. Data Mining (ICDM), Nov. 2017, pp. 1–10.
[3] V. Botta-Genoulaz, P.-A. Millet, and B. Grabot, “A survey on the recent research literature on ERP systems,” Comput. Ind., vol. 56, no. 6, pp. 510–522, Aug. 2005.
[4] P. P. S. Chen, “The entity-relationship model—Toward a unified view of data,” ACM Trans. Database Syst., vol. 1, no. 1, pp. 9–36, 1976.
[5] D. Cohn and R. Hull, “Business artifacts: A data-centric approach to modeling business operations and processes,” IEEE Data Eng. Bull., vol. 32, no. 3, pp. 3–9, Sep. 2009.
[6] R. Conforti, M. Dumas, L. García-Bañuelos, and M. L. Rosa, “BPMN Miner: Automated discovery of BPMN process models with hierarchical structure,” Inf. Syst., vol. 56, pp. 284–303, Mar. 2016.
[7] M. de Leoni and W. M. P. van der Aalst, “Data-aware process mining: Discovering decisions in processes using alignments,” in *Proc. 28th Annu. ACM Symp. Appl. Comput. (SAC)*, 2013, pp. 1454–1461.

[8] H. J. Genrich and K. Lautenbach, “The analysis of distributed systems by means of predicate/transition-nets,” in *Semantics of Concurrent Computation (Lecture Notes in Computer Science)*, vol. 70, G. Kahn, Ed. Berlin, Germany: Springer, 1979, pp. 123–146.

[9] H. J. Genrich, “Predicate/transition-nets,” in *Petri Nets: Central Models and Their Properties* (Lecture Notes in Computer Science), vol. 254, W. Brauer, W. Reisig, and G. Rozenberg, Eds. Berlin, Germany: Springer, 1987, pp. 207–247.

[10] OMG Unified Modeling Language 2.5. OMG, Object Manag. Group, Needham, MA, USA, 2013.

[11] T. Halpin and T. Morgan, *Information Modeling and Relational Databases*. San Francisco, CA, USA: Morgan Kaufmann, 2008.

[12] R. Hull, E. Damaggio, R. De Masellis, F. Fournier, M. Gupta, F. T. Heath, S. Hobson, M. Linehan, S. Maradugu, A. Nigam, and P. N. Sukaviriya, “Business artifacts with guard-stage-milestone lifecycles: Managing artifact interactions with conditions and events,” in *Proc. Int. Conf. Distrib. Event-Based Syst.*, 2011, pp. 51–62.

[13] K. Jensen, “Coloured Petri nets,” in *Petri Nets: Central Models and Their Properties* (Lecture Notes in Computer Science), vol. 254, W. Brauer, W. Reisig, and G. Rozenberg, Eds. Berlin, Germany: Springer, 1987, pp. 248–299.

[14] K. Jensen, *Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use*. Berlin, Germany: Springer, 1996.

[15] S. J. J. Leemans, D. Fahland, and W. M. P. van der Aalst, “Discovering block-structured process models from event logs—A constructive approach,” in *Proc. Int. Conf. Appl. Theory Petri Nets Concurrence*. Berlin, Germany: Springer, 2013, pp. 311–329.

[16] S. J. Leemans, D. Fahland, and W. M. P. van der Aalst, “Process and deviation exploration with inductive visual miner,” *BPM (Demos)*, vol. 1295, p. 46, Sep. 2014.

[17] G. Li, R. M. de Carvalho, and W. M. P. van der Aalst, *Automatic Discovery of Object-Centric Behavioral Constraint Models*. Cham, Switzerland: Springer, 2017, pp. 43–58.

[18] N. Lohmann, “Compliance by design for artifact-centric business processes,” in *Business Process Management* (Lecture Notes in Computer Science), vol. 6896. Berlin, Germany: Springer, 2011, pp. 99–115.

[19] X. Lu, M. Nagelkerke, D. van de Wiel, and D. Fahland, “Discovering interacting artifacts from ERP systems,” *IEEE Trans. Serv. Comput.*, vol. 8, no. 6, pp. 861–873, Dec. 2015.

[20] F. M. Maggi, M. Dumas, L. García-Bañuelos, and M. Montali, “Discovering data-aware declarative process models from event logs,” in *Business Process Management* (Lecture Notes in Computer Science). Berlin, Germany: Springer, 2013, pp. 81–96.

[21] F. M. Maggi, A. J. Mooij, and W. M. P. van der Aalst, “User-guided discovery of declarative process models,” in *Proc. IEEE Symp. Comput. Intell. Data Mining (CIDM)*, Apr. 2011, pp. 192–199.

[22] M. Montali, F. Chesani, P. Mello, and F. M. Maggi, “Towards data-aware constraints in declare,” in *Proc. 28th Annu. ACM Symp. Appl. Comput. (SAC)*, 2013, pp. 1391–1396.

[23] Y. B. Moon, “Enterprise resource planning (ERP): A review of the literature,” *Int. J. Manag. Enterprise Develop.*, vol. 4, no. 3, pp. 235–264, 2007.

[24] A. Rozinat and W. M. P. van der Aalst, “Decision mining in ProM,” in *Proc. Int. Conf. Bus. Process Manag.*. Berlin, Germany: Springer, 2006, pp. 420–425.

[25] H. Selig, “Continuous event log extraction for process mining,” M.S. thesis, Dept. Comput. Sci., KTH Roy. Inst. Technol., Stockholm, Sweden, 2017.