Automatic Discovery of Object-Centric Behavioral Constraint Models

Guangming Li, Renata Medeiros de Carvalho, and Wil M.P. van der Aalst

Eindhoven University of Technology, P.O. Box 513, 5600 MB, Eindhoven, The Netherlands.
g.li.3@tue.nl, r.carvalho@tue.nl, w.m.p.v.d.aalst@tue.nl

Abstract. Process discovery techniques have successfully been applied in a range of domains to automatically discover process models from event data. Unfortunately existing discovery techniques only discover a behavioral perspective of processes, where the data perspective is often as a second-class citizen. Besides, these discovery techniques fail to deal with object-centric data with many-to-many relationships. Therefore, in this paper, we aim to discover a novel modeling language which combines data models with declarative models, and the resulting *object-centric behavioral constraint model* is able to describe processes involving *interacting instances* and *complex data dependencies*. Moreover we propose an algorithm to discover such models.

Keywords: Process mining · Object-centric modeling · Process discovery · Cardinality constraints

1 Introduction

Process discovery is one of the most challenging process mining tasks. However, state of the art techniques can already deal with situations where each process instance is recorded as a case with ordered events and each event is related to exactly one case by a case identifier [1]. Examples of algorithms that consider process instances to derive models include the Inductive Miner, ILP Miner, Heuristic Miner and Declare Miner, distributed as ProM plugins.[1] All examples extract models from behavior-centric logs (e.g., XES logs). Moreover, there are already over 20 commercial software products supporting process mining (e.g., Disco, Celonis, ProcessGold, QPR, etc.).

However, when it comes to data-centric/object-centric processes supported by CRM and ERP systems, most of the existing discovery techniques fail. Such systems have one-to-many and many-to-many relationships between data objects that makes it impossible to identify a unique process instance notion to group traces. If we enforce such a grouping anyway, it leads to convergence and divergence problems. Besides, the discovered models using existing approaches are often based on business process modeling languages such as Petri nets, BPMN diagrams, Workflow nets, EPCs, and UML activity diagrams. They typically consider process instances in isolation, ignoring interactions in between. Moreover, they cannot model the data perspective in a precise manner. Data objects can be modeled, but the more powerful constructs (e.g., cardinality constraints)

[1] http://www.processmining.org/prom/start

used in Entity-Relationship (ER) models [5], UML class models [9] and Object-Role Models (ORM) [10] cannot be reflected at all in today's process models. As a result, data and control-flow need to be described in separate diagrams.

Numerous approaches in literature tried to solve the problems mentioned above. Various techniques of *colored Petri nets*, i.e., Petri nets where tokens have a value, are employed to add data to process models [23, 8, 7, 13, 14]. These approaches do not support explicit data modeling, i.e., there is no data model to relate entities and activities. The earliest approaches that explicitly related process models and data models were proposed in the 1990s [22, 11]. One example is the approach by Kees van Hee [11] who combined Petri nets, a specification, and a binary data model. Other approaches such as data-aware process mining discovery techniques [17, 21] extend the control-flow perspective with the data perspective. They discover the control-flow perspective of processes, using one of the process discovery techniques available today (e.g., inductive mining techniques), and then the data perspective (e.g., read and write operations, decision points and transition guards) using standard data mining techniques. These techniques mainly focus on control-flow perspective, considering the data perspective as a second-class citizen. Artifact-centric approaches [6, 12, 15, 18] (including the work on proclets [2]) attempt to describe business processes in terms of so-called business artifacts. Artifacts have data and lifecycles attached to them, thus relating both perspectives. There are a few approaches to discover artifact-centric models from data-centric processes [19, 16, 20]. However, these force users to specify artifacts as well as a single instance notion within each artifact, and tend to result in complex specifications that are not fully graphical and distribute the different instance types over multiple diagrams.

This paper uses a novel modeling language, named *Object-Centric Behavioral Constraint (OCBC)*, that combines declarative language (*Declare* [4]), and data/object modeling techniques (ER, UML, or ORM) [3]. Cardinality constrains are used as a unifying mechanism to tackle data and behavioral dependencies, as well as their interplay. Besides motivating that the novel language is useful for modeling data-centric processes, we also propose an algorithm for discovering OCBC models from event data lacking a clear process instance notion. By doing this, we demonstrate that this novel modeling language has potential to be used as an alternative to mainstream languages for all kinds of process mining applications.

The remainder is organized as follows. Section 2 presents a process to introduce OCBC models. Section 3 illustrates the ingredients of OCBC models. Our discovery algorithm is proposed in Section 4. Section 5 shows some experimental results showing the validity of our approach and implementation and Section 6 concludes the paper.

2 Motivation Example

In this section, the Order To Cash (OTC) process, which is the most typical business process supported by an ERP system, is employed to illustrate OCBC models. The OTC process has many variants and our example is based on the scenario in *Dolibarr*. [2]

[2] 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.

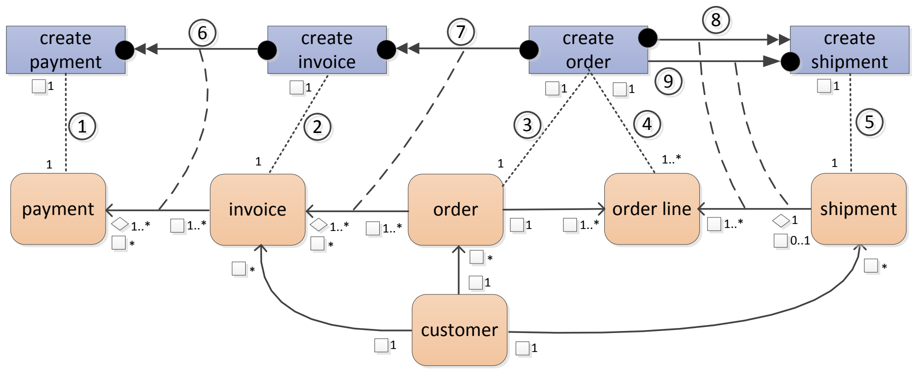

Fig. 1. A small *Object-Centric Behavioral Constraint* (OCBC) model.

Figure 1 shows an OCBC model which describes the OTC process in Dolibarr. The top part shows behavioral constraints. These describe the ordering of activities (*create order*, *create invoice*, *create payment*, and *create shipment*). The bottom part describes the structuring of objects relevant for the process, which can be read as if it was a UML class diagram (with six object classes *order*, *order line*, *invoice*, *payment*, *shipment*, and *customer*). Note that an order has at least one order line, each order line corresponds to precisely one shipment, each order refers to one or more invoices, each invoice refers to one or more payments, each order, shipment or invoice refers to one customer, etc. The middle part relates activities, constraints, and classes.

The notation will be explained in more detail later. However, to introduce the main concepts, we first informally describe the 9 constructs highlighted in Figure 1. Construct (3) indicates a one-to-one correspondence between *order* objects and *create order* events. If an object is added to the class *order*, the corresponding activity needs to be executed and vice versa. (1), (2) and (5) also represent the one-to-one correspondence. (4) shows a one-to-many relation between *create order* events and *order line* objects. (6) expresses that each *create invoice* event is followed by one or more corresponding *create payment* events and each *create payment* activity is preceded by one or more corresponding *create invoice* events. A similar constraint is expressed by (7). (8) demands that each *create order* event is followed by at least one corresponding *create shipment* event. (9) denotes that each *create shipment* event is preceded by precisely one corresponding *create order* event. Note that *one payment* can cover *multiple invoices* and *multiple payments* can be executed for *a particular invoice* (i.e., one payment only covers a part of the invoice). Obviously, this process has one-to-many and many-to-many relations, and it is impossible to identify a single case notion.

The process described in Figure 1 cannot be modeled using conventional notations (e.g., BPMN) because (a) four different types of instances are intertwined and (b) constraints in the class model influence the allowed behavior. Moreover, the OCBC model provides a *full* specification of the allowed behavior in a *single diagram*, so that no further coding or annotation is needed.

3 Object-Centric Behavioral Constraint (OCBC) Modeling Language

After introducing OCBC models based on a typical real-life process, we describe the data perspective and the behavioral perspective, and show how OCBC models relate both perspectives. See [3] for the formal definition of the OCBC language.

3.1 Modeling Data Cardinality Constraints

In this paper, the term "object" is different from it used in other fields, such as software engineering. In general, objects are data elements generated and used by information systems. These are grouped in classes and have some attributes. For example, a record in the "order" table can be considered as an object of class "order". Each value (e.g., a customer name "Mary") in the record can be considered as an attribute of the object.

Cardinalities indicates non-empty sets of integers, i.e., "1..*" denotes the set of positive integers $\{1, 2, ...\}$. Objects may be related and cardinality constraints help to structure dependencies. As shown in Figure 2(a), we use a subset of mainstream notations to specify a *class model* with temporal annotations such as "eventually" cardinalities (indicated by $\Diamond$) and "always" cardinalities (indicated by $\Box$).[3]

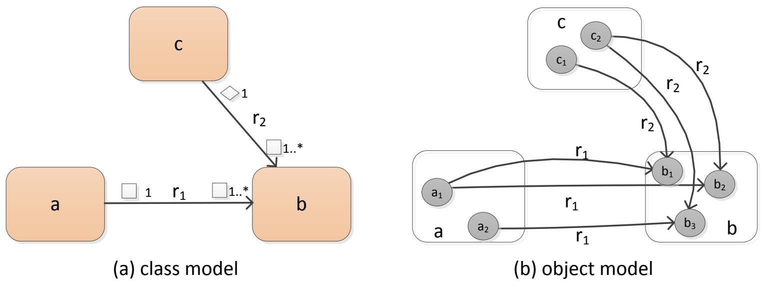

Fig. 2. Example of a class model and corresponding object model.

A class model contains a set of object classes (OC) and a set of relationship types (RT). Relationship types are directed (starting from source classes and pointing to target classes) and each one defines two cardinality constraints: one on its source side (close to the source class) and one on its target side (close to the target class).[4]

The class model depicted in Figure 2(a) has three object classes, i.e., $OC = \{a, b, c\}$ and two relationship types, i.e., $RT = \{r_1, r_2\}$. r_1 points to b from a, which indicates a is the source class, b is the target class, and a and b are related through r_1.

[3] $\Box$ indicates the constraint should hold at any point in time and $\Diamond$ indicates the constraint should hold from some point onwards.

[4] For the sake of brevity, we omit redundant cardinalities in the graph. For instance, "$\Box$1" implies "$\Diamond$1" and therefore "$\Diamond$1" can be removed in this case.

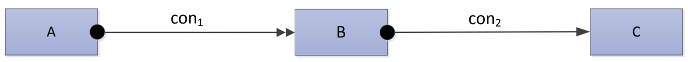

Fig. 3. An example behavioral model with two behavioral cardinality constraints.

The annotation "$\Box 1..*$" on the target side of r_1 indicates that for each object in a, there is always at least one corresponding object in b. "$\Diamond 1$" on the source side of r_2 indicates that for each object in b, there is eventually precisely one corresponding object in c. A class model defines a "space" of possible *object models*, i.e., concrete collections of objects and relations instantiating the class model.

An object model includes a set of objects (Obj) and a set of object relations (Rel). More precisely, an object relation can be viewed as a tuple consisting of a class relationship type, a source object and a target object. For instance, (r_1, a_1, b_1) is an object relation, with r_1 as its name, a_1 as the source object, b_1 as the target object, and a_1 and b_1 are related through r_1. Note that each object has a corresponding object class, e.g., a_1 corresponds to the object class a.

Figure 2(b) shows an object model. The objects are depicted as grey dots: $Obj = \{a_1, a_2, b_1, b_2, b_3, c_1, c_2\}$. Among them, a_1 and a_2 belong to object class a; b_1, b_2 and b_3 belong to object class b; c_1 and c_2 belong to object class c. There are three relations corresponding to relationship r_1 (e.g., (r_1, a_1, b_1)), and three relations corresponding to relationship r_2 (e.g., (r_2, c_1, b_1)).

3.2 Modeling Behavioral Cardinality Constraints

A process model can be viewed as a set of *constraints*. For example, in a procedural language like Petri nets, places correspond to constraints: removing a place may allow for more behavior and adding a place can only restrict behavior. In this paper, we will employ a graphical notation inspired by *Declare*, a declarative workflow language [4].

Figure 3 shows two example behavioral constraints: con_1 and con_2. Each constraint corresponds to one constraint type. Table 1 shows eight examples of constraint types. Constraint con_1 is a *response* constraint and constraint con_2 is a *unary-response* constraint. The graphical representations of the eight example constraint types are shown in Figure 4. Besides the example constraint types, we allow for any constraint type that can be specified in terms of the *cardinality of preceding and succeeding target events relative to a collection of reference events*. As a shorthand, one arrow may combine two constraints as shown in Figure 5. For example, constraint con_{56} states that after creating an order there is precisely one validation and before a validation there is precisely one order creation.

Given some *reference event* e we can reason about the events *before* e and the events *after* e. One constraint type may require that the number of corresponding events of one particular reference event before or after the event lies within a particular range (e.g., before$\geqslant$ 0 and after $\geqslant$ 1 for *response*). For instance, constraint con_1 specifies that each A event should be succeeded by at least one corresponding B event and constraint con_2 specifies that each B event should be succeeded by precisely one C event.

Table 1. Examples of constraint types, inspired by *Declare*. Note that a constraint is defined with respect of a reference event.

constraint	formalization
response	$\{(\mathit{before}, \mathit{after}) \in \mathbb{N} \times \mathbb{N} \mid \mathit{after} \geq 1\}$
unary-response	$\{(\mathit{before}, \mathit{after}) \in \mathbb{N} \times \mathbb{N} \mid \mathit{after} = 1\}$
non-response	$\{(\mathit{before}, \mathit{after}) \in \mathbb{N} \times \mathbb{N} \mid \mathit{after} = 0\}$
precedence	$\{(\mathit{before}, \mathit{after}) \in \mathbb{N} \times \mathbb{N} \mid \mathit{before} \geq 1\}$
unary-precedence	$\{(\mathit{before}, \mathit{after}) \in \mathbb{N} \times \mathbb{N} \mid \mathit{before} = 1\}$
non-precedence	$\{(\mathit{before}, \mathit{after}) \in \mathbb{N} \times \mathbb{N} \mid \mathit{before} = 0\}$
co-existence	$\{(\mathit{before}, \mathit{after}) \in \mathbb{N} \times \mathbb{N} \mid \mathit{before} + \mathit{after} \geq 1\}$
non-co-existence	$\{(\mathit{before}, \mathit{after}) \in \mathbb{N} \times \mathbb{N} \mid \mathit{before} + \mathit{after} = 0\}$

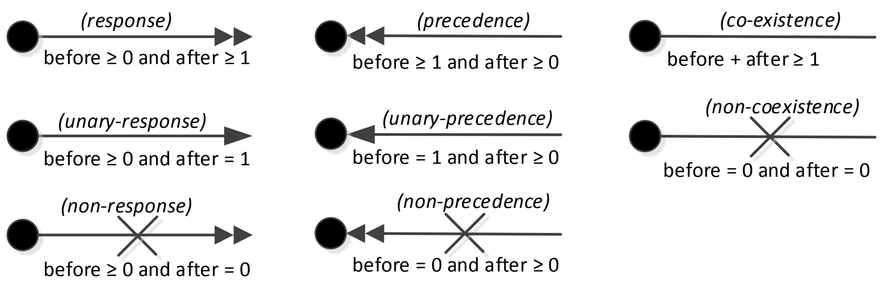

Fig. 4. Graphical notation for the example constraint types defined in Table 1. The dot on the left-hand side of each constraint refers to the *reference events*. *Target events* are on the other side that has no dot. The notation is inspired by *Declare*, but formalized in terms of cardinality constraints rather than LTL.

A *behavioral constraint model* is a collection of activities and constraints. More precisely, a constraint corresponds to a constraint type, a reference activity and a target activity. Figure 3 displays a behavioral model consisting of two constraints (con_1 and con_2) and three activities (A, B and C). Each constraint has a dot referring to the *reference activity*. The corresponding *target activity* can be found on the other side. For example, the reference activity of con_2 is B (see dot) and the target activity of con_2 is C. The shape (e.g., a double-headed arrow) of each constraint indicates the constraint type. For instance, con_1 has a dot on the left side and a double-headed arrow on the right side, which means the corresponding constraint type is *response*, the reference activity is A and the target activity is B.

3.3 Object-Centric Behavioral Constraints

Section 3.1 focused on structuring objects and formalizing cardinality constraints on object models (i.e., classical data modeling) while Section 3.2 focused on control-flow modeling and formalizing behavioral constraints *without* considering the structure of objects. This subsection relates *both perspectives* by combining control-flow modeling and data modeling to fully address the challenges described in the introduction.

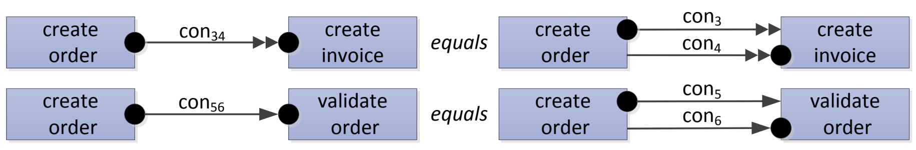

Fig. 5. An arrow with two reference events (●) can be used as a shorthand. Constraint con_{34} (con_{56}) corresponds to the conjunction of constraints con_3 and con_4 (resp. con_5 and con_6).

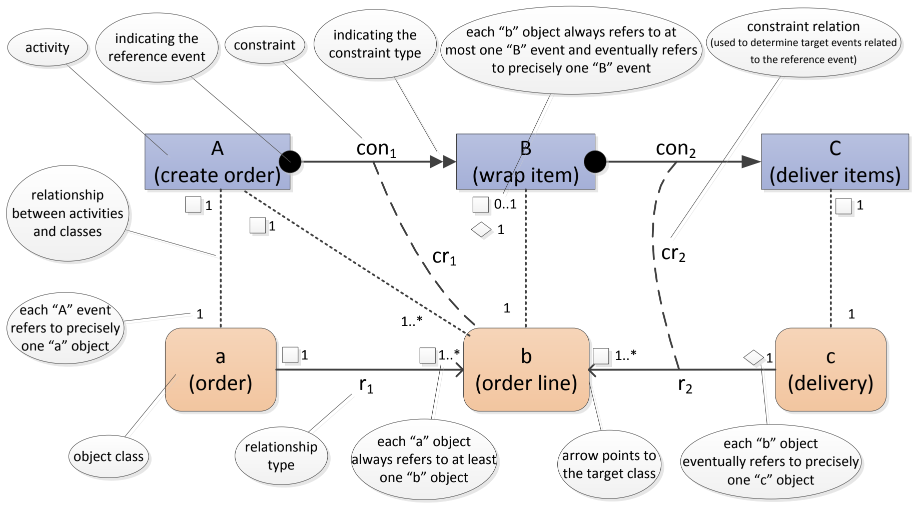
activity
indicating the reference event
constraint
indicating the constraint type
each "b" object always refers to at most one "B" event and eventually refers to precisely one "B" event
constraint relation (used to determine target events related to the reference event)
A (create order)
con1
B (wrap item)
con2
C (deliver items)
relationship between activities and classes
each "A" event refers to precisely one "a" object
cr1
cr2
0..1
1
1..*
a (order)
b (order line)
c (delivery)
r1
r2
object class
relationship type
each "a" object always refers to at least one "b" object
arrow points to the target class
each "b" object eventually refers to precisely one "c" object

Fig. 6. An example model illustrating the main ingredients of OCBC models.

We use so-called AOC relationships (denoted by a dotted line between activities and classes) and constraint relations (denoted by a dashed line between behavioral constraints and classes or class relationships) to combine the behavioral constraint model in Figure 3 with the class model in Figure 2, resulting in the complete example OCBC model in Figure 6. For better understanding, we attach a scenario on the model. For example, activity A corresponds to *create order* activity while class a corresponds to class *order*.

The example model has four AOC relationships, i.e., $AOC = \{(A, a), (A, b), (B, b), (C, c)\}$.[5] Note that A refers to object classes a and b while b refers to activities A and B. This shows that OCBC models are capable of modeling one-to-many and many-to-many relationships between events and objects. AOC relationships also have cardinalities. The $\Box$ ($\Diamond$) cardinalities on the activity side define how many events there always (eventually) need to be for each object. The cardinalities on the class side

[5] In this paper, we use the upper-case (lower-case) letters to express activities (classes), and use the upper-case (lower-case) letters with a footnote to express events (objects).

(without $\Box$ or $\Diamond$ symbols) define how many objects there need to be for each event when the event occurs.

Constraint relations define the *scope* of each constraint thereby relating reference events to selected target events. If a constraint relation connects a constraint to a class, events are correlated through objects of this class. Consider the constraint relation cr_1 between con_1 and b. Let A_1 be one reference event for con_1 (i.e., one *create order* event) and A_1 refers to a set of b objects (i.e., *order line* objects). Each B event (i.e., *wrap item* event) that refers to at least one object in the set is the target event of A_1 for cr_1 . If a constraint relation connects a constraint to a relationship, the target events are related to the reference event through object relations (of this relationship) in the object model. Consider the constraint relation cr_2 between con_2 and r_2. Let B_1 be one reference event for con_2 (i.e., one *wrap item* event) and B_1 refers to b objects (i.e., *order line* objects) which are related to c object (i.e., *delivery* objects) through r_2 relations. Each C event (i.e., *deliver items* event) that refers to at least one one of these c objects (i.e., *delivery* objects) is the target event of B_1 for cr_2. Note that, indicated by the example model, B_1 refers to precisely one b object that is related to one c object, which means B_1 has precisely one target event.

4 Discovery of Object-Centric Behavioral Models

In this section, we specify a new format of logs that are object-centric, and propose a novel algorithm to discover OCBC models based on such logs.

4.1 Object-Centric Event Logs

A process is merely a collection of *events* without assuming some case or process instance notion, and the corresponding event log provides a *snapshot of the object model after each event*, where the object model represents the state of the process. Such a log can be extracted from real-life IT systems. For instance, the Oracle database provides *change tables* to record any modification in the database. With these tables, it is possible to reconstruct any previous state of the database. Besides, without the change tables, it is still possible to produce such a log by exploiting explicit change logs in systems like SAP.

In a log, each event corresponds to an object model (in the "Object Model" column) which represents the state of the process just after the execution of the event. Besides, each event corresponds to an *activity* and may have additional *attributes*, e.g., the time at which the event took place. Moreover, events are atomic and ordered (indicated by the "Index" column). In order to relate the behavioral perspective and the data perspective (i.e., events and objects), each event also refers to at least one object (in the "Reference" column). Logs of this format are called object-centric event logs (denoted as *XOC* logs in remainder).

Table 2 gives an example XOC log containing 7 events. Event A_1 corresponds to the first occurrence of activity A, has one attribute att_1 whose value is v_1 and refers to three objects: a_1, b_1, and b_2. The corresponding object model of A_1 consists of three objects and two object relations. Table 2 also illustrates the evolution of the object model. After the occurrence of some event, objects may have been added, and relations may have been added or removed.[6] Note that the example log has the same scenario as indicated by the model in Figure 6, e.g., activity A means activity *create order* and object a_1 means an *order* object.

Table 2. An example XOC log

Index	Event	Activity	Attributes	References	Object Model	
					Objects	Relations
1	A_1	A	$\{att_1 = v_1\}$	$\{a_1, b_1, b_2\}$	$\{a_1, b_1, b_2\}$	$\{(r_1, a_1, b_1), (r_1, a_1, b_2)\}$
2	B_1	B	$\{att_2 = v_2\}$	$\{b_1\}$	$\{a_1, b_1, b_2\}$	$\{(r_1, a_1, b_1), (r_1, a_1, b_2)\}$
3	B_2	B	$\{att_2 = v_3\}$	$\{b_2\}$	$\{a_1, b_1, b_2\}$	$\{(r_1, a_1, b_1), (r_1, a_1, b_2)\}$
4	A_2	A	$\{att_1 = v_4\}$	$\{a_2, b_3\}$	$\{a_1, a_2, b_1, b_2, b_3\}$	$\{(r_1, a_1, b_1), (r_1, a_1, b_2), (r_1, a_2, b_3)\}$
5	B_3	B	$\{att_2 = v_5\}$	$\{b_3\}$	$\{a_1, a_2, b_1, b_2, b_3\}$	$\{(r_1, a_1, b_1), (r_1, a_1, b_2), (r_1, a_2, b_3)\}$
6	C_1	C	$\{att_3 = v_6, att_4 = v_7\}$	$\{c_1\}$	$\{a_1, a_2, b_1, b_2, b_3, c_1\}$	$\{(r_1, a_1, b_1), (r_1, a_1, b_2), (r_1, a_2, b_3), (r_2, c_1, b_1)\}$
7	C_2	C	$\{att_3 = v_8, att_4 = v_9\}$	$\{c_2\}$	$\{a_1, a_2, b_1, b_2, b_3, c_1, c_2\}$	$\{(r_1, a_1, b_1), (r_1, a_1, b_2), (r_1, a_2, b_3), (r_2, c_1, b_1), (r_2, c_2, b_2), (r_2, c_2, b_3)\}$

4.2 Discovery Algorithm

The algorithm takes an XOC log as well as a set of possible behavioral constraint types as input, which means users can specify the constraint type set based on their needs. In Figure 3 the *response* and *unary-response* types were used, but the user can select from a range of possible types that can be discovered. Next, we explain the discovery process based on the example log.

4.2.1 Discovery of Class Models

In general, the class model is discovered based on the object models in the input log. Figure 2(a) shows the discovered class model from the example log, where $OC = \{a, b, c\}$ and $RT = \{r_1, r_2\}$.

OC can be learned by incorporating all classes of all objects in the object models of all events. For instance, a is a discovered class since object models contain objects of class a, e.g., a_1. RT can be learned through observing object relations in object models of each event. r_1 (having a as the source class and b as the target class) is discovered since there exist object relations involving r_1, e.g., (r_1, a_1, b_1), and each of them has a object as the source object and b object as the target object.

[6] We assume that objects cannot change class or be removed at a later stage to avoid referencing non-existent objects. Objects can be marked as deleted but cannot be removed (e.g, by using an attribute or relation).

For each relationship, its "always" ("eventually") cardinalities can be derived through integrating the number of related objects of each reference object in the object model of each (the last) event.[7] For instance, the discovered "always" cardinality on the source side of r_1 is "1" since in the object model of each event, each b object has precisely one related a object, e.g., b_1 and b_2 have one related a object a_1. The discovered "eventually" cardinality on the source side of r_1 is also "1" since in the object model of the last event (i.e., C_2), b_1 and b_2 have one related a object a_1 while b_3 has one related a object a_2 (the "eventually" cardinality is omitted on the graph for simplicity).

Note that the directly discovered "always" and "eventually" cardinalities on the target side of r_1 should be $\{1,2\}$, since a_1 has two related b objects (b_1 and b_2) while a_2 has one related b object (b_3). We use a strategy to extend $\{1,2\}$ to $\{1,2,...\}$, which will be explained later.

4.2.2 Discovery of AOC Relationships

After the class model is discovered, we can mine AOC relationships based on the objects referred to by each event.[8] The idea is that if an event refers to an object, the activity of the event refers to the class of the object. For instance, since event A_1 refers to three objects a_1, b_1, and b_2, activity A refers to class a and b, which means two AOC relationship (A,a) and (A,b) can be discovered as shown in Figure 6.

For each AOC relationship, its cardinalities on the class side can be achieved by incorporating numbers of referred objects by each event. Consider the cardinality on the class side of (A,b). Since A_1 has two referred b objects (b_1 and b_2) while A_2 has one referred b object (b_3), the directly discovered cardinality is $\{1,2\}$ and it is extended to $\{1,2,...\}$. Similarly, the "always" ("eventually") cardinalities on the activity side can be achieved by incorporating numbers of events referring each reference object just after every (the last) event happens. Consider the cardinality on the activity side of (B,b). Since b_1 and b_2 are not referred by any B event after the first event A_1 just happens, 0 is an element of the "always" cardinality. After the second event B_1 just happens, b_1 is referred by B_1, which adds a new element "1" into the "always" cardinality. After we check all events, the discovered "always" cardinality is $\{0,1\}$. In terms of the "eventually" cardinality on the activity side of (B,b), we just check the moment when the last event just happens. Since each b object is referred by precisely one B event (i.e., b_1 is referred by B_1, b_2 is referred by B_2 and b_3 is referred by B_3), the discovered "eventually" cardinality is $\{1\}$.

4.2.3 Discovery of Behavioral Models

Based on the discovered class model and AOC relationships, we can relate events by

[7] In terms of cardinalities on the source (target) side of a relationship, the objects in the target (source) class are reference objects.

[8] There is a reference relation between an event (e.g., A_1) and an object (e.g., a_1) if and only if the event refers to the object, denoted as (A_1,a_1). The reference relations accumulate along with the occurrence of events. For instance, after A_1 happens, the set of reference relations is $\{(A_1,a_1),(A_1,b_1),(A_1,b_2)\}$; after B_1 happens, the set of reference relations is $\{(A_1,a_1),(A_1,b_1),(A_1,b_2),(B_1,b_1)\}$

objects and discover the constraints between activities. More precisely, each pair of activities referring to the same class or two related classes may have potential constraints in between. The class or the relationship between the two related classes serves as the intermediary to relate events. Note that each potential constraint, e.g., con, between an activity pair, e.g., (A, B), takes A as the reference activity and B as the target activity, and corresponds to a constraint relation which connects the constraint and its intermediary. The constraint relation can identify the target events of each reference event for con (cf. Section 3.3). If the relation between each reference event and its target events satisfies the restriction indicated by a constraint type, e.g., *response* (cf. Section 3.2), the potential constraint con becomes a discovered constraint which takes *response* as the constraint type. Consider the activities A and B in Figure 6 (assuming the model does not have behavioral constraints) and the example log. Since both A and B refer to b, they have potential constraints in-between. If we assume A is the reference activity, then there are two reference events A_1 and A_2, where A_1 is followed by two target events B_1 and B_2, and A_2 is followed by one target event B_3. This relation satisfies the requirement indicated by constraint type *response*, resulting in a discovered constraint con_1.

4.3 Discussion of Model Metrics

Based on the above three steps, we can discover a model similar to the one shown in Figure 6 from the example log. The fitness of the discovered model is 1. As we mentioned, we need heuristics to extend the directly discovered cardinalities, e.g., when to replace $\{1, 2, 5, 8\}$ by $1..*$? Since the directly discovered cardinalities only contain the actual numbers observed in the log, their quality depends on the size of the log, i.e., if the log is not large enough to contain complete cardinalities in the process, the discovered model is overfitting. In order to improve generalization, we can extend cardinalities to allow more possibilities. An extreme example is to extend all directly discovered cardinalities to "$*$", which allows all possibilities and makes the model to be underfitting.

The difference between the discovered model and the one shown in Figure 6 is that the former one has more behavioral constraints (e.g., a constraint with B as its reference activity and A as its target activity). In this sense, discovered models tend to have too many behavioral constraints, since our algorithm discovers all allowed constraints between each activity pair. This often makes discovered models spaghetti-like. In order to get more understandable models, we can remove less important constraints based on the specific situation. For instance, implied constraints can be removed without losing fitness and precision.[9] Note that, in general, filtering a model tends to improve (at least remain) fitness (i.e., more behavior fits the model), decrease complexity (i.e., the model has fewer edges), improve generalization (i.e., more behavior is allowed) and degrades precision (i.e., unobserved behaviors in the log may become allowed). Based on the specific need, one needs to balance between such concerns. Our plugin introduced in next section allows for seamless navigation possibilities to balance fitness, precision and simplicity.

[9] The implied constraint by one constraint has the same reference activity, the same target activity and refers to the same class or relationship as the constraint as well as allowing more behavior than the constraint.

5 Experiments

The discovery algorithm was validated based on logs extracted from data generated by the Dolibarr ERP/CRM system when executing the OTC (Order to Cash) process. More precisely, the data was extracted from 6 tables in the database of Dolibarr. For instance, "llx_commande" table records customer orders while "llx_facture" table consists of invoices. Based on the tables, we derived 4 activities (*create order*, *create invoice*, *create shipment* and *create payment*) and 6 object classes (i.e., one table corresponds to one object class) to be included in the XOC logs.[10] We instrumented the ERP/CRM system in such a way that we could extract data executed by real and simulated users of the system.

Our algorithm has been implemented in the "OCBC Model Discovery" Plugin in ProM. [11] Figure 7 shows the interface of the plugin and a discovered model (in panel ④) from an XOC log . 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 metrics of constraints such as confidence and support (this discussion is beyond the paper). It is possible to zoom in/out models through operating panel ③.

As discussed in last section, we can filter the discovered models to get a better understanding. Using the filter panels, it is possible to filter behavioral constraints based on constraint types (the plugin discovers all constraints of 9 common types by default) and activity names through panel ⑤, or based on the regulation of fitness and precision (the method for computing fitness and precision is not covered by this paper) through panel ⑥. For instance, if the desired action is to inspect the *unary-response*, *response*, *unary-precedence* and *precedence* constraints between *create order* and *create shipment* activities, one can uncheck the other boxes (all boxes are checked by default). The filtered model for the example log is shown in Figure 8.

In the filtered model, there exist a *response* and a *unary-precedence* constraints between *create order* and *create shipment* activities. The constraints indicate one *create order* event is followed by one or more corresponding *create shipment* events while one *create shipment* event is always preceded by precisely one corresponding *create order* event. Investigating the Dolibarr system and its tables, it is possible to affirm that the process behavior of the system is according to these statements. The system allows creating multiple shipments for one order, but does not allow a shipment to contain products from multiple orders (as shown in Figure 8). Although the discovered model in Figure 7 is more complex than the real model (designed based on the real process) in Figure 1, we can easily get the same insights after filtering appropriately.

6 Conclusion

In this paper we introduced *Object-Centric Behavioral Constraint* (OCBC) modeling language to graphically model control-flow and data/objects in a truly integrated man-

[10] These tables and logs can be found at https://svn.win.tue.nl/repos/prom/Packages/OCBC/Trunk/tests/testfiles/logs&models/OCBCModelDiscovery.

[11] Download *ProM 6 Nightly builds* from http://www.promtools.org/prom6/nightly/ and update the *OCBC package*.

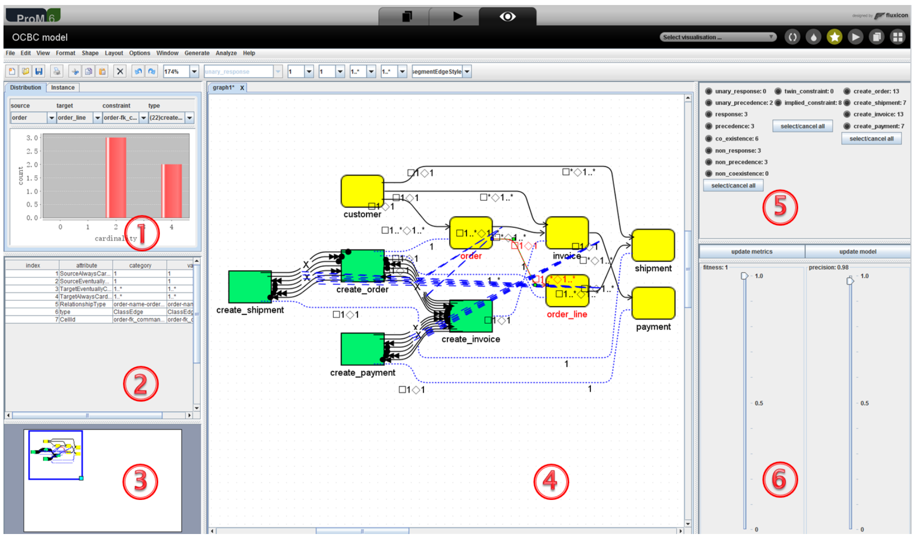

Fig. 7. The interface of the "OCBC Model Discovery" Plugin.

ner. This novel language uses cardinality constraints to *describe data and behavioral perspectives in a single diagram* which overcomes the problems of existing data-aware approaches that separate the data (e.g., a class model) and behavioral (e.g., BPMN, EPCs, or Petri nets) perspectives. In OCBC models, different types of instances can interact in a fine-grained manner and the constraints in the class model guide behavior.

In this paper, we proposed an algorithm to discover OCBC models from object-centric event logs. Currently, the discovered models perfectly fit the source logs (i.e., there is no noise in logs or we do not distinguish noise). In future, we will extend the algorithm to better deal with infrequent and incomplete behavior. Besides, some metrics such as fitness, precision and generalization will be proposed to evaluate discovered models. Also, we will improve our approach to deal with larger scale logs in more complex scenarios, i.e., enabling the approach to discover compact models in a scalable manner (e.g., remove redundancies).

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. W.M.P. van der Aalst. *Process Mining: Data Science in Action*. Springer-Verlag, 2016.
2. W.M.P. van der Aalst, P. Barthelmess, C.A. Ellis, and J. Wainer. Proclets: A Framework for Lightweight Interacting Workflow Processes. *International Journal of Cooperative Information Systems*, 10(4), 2001.

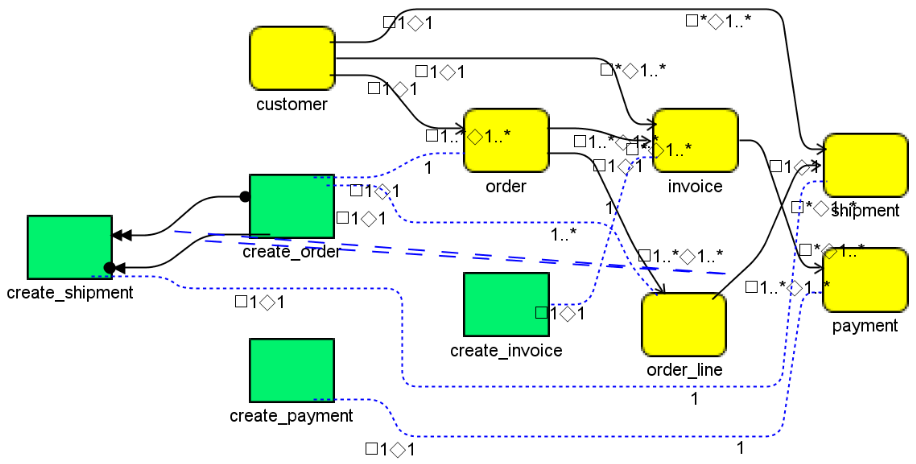

Fig. 8. The model discovered from the OTC process after filtering.

3. W.M.P. van der Aalst, G. Li, and M. Marco. Object-Centric Behavioral Constraints. Corr technical report, arXiv.org e-Print archive, 2017. Available at https://arxiv.org/abs/1703.05740.
4. W.M.P. van der Aalst, M. Pesic, and H. Schonenberg. Declarative Workflows: Balancing Between Flexibility and Support. *Computer Science - Research and Development*, 23(2), 2009.
5. P. Chen. The Entity-Relationship Model - Toward a Unified View of Data. *ACM Transaction on Database Systems*, 1(1), 1976.
6. D. Cohn and R. Hull. Business Artifacts: A Data-Centric Approach to Modeling Business Operations and Processes. *IEEE Data Engineering Bulletin*, 32(3), 2009.
7. H.J. Genrich. Predicate/Transition-Nets. In W. Brauer, W. Reisig, and G. Rozenberg, editors, *Advances in Petri Nets 1986 Part I: Petri Nets, Central Models and Their Properties*, volume 254 of *Lecture Notes in Computer Science*, pages 207–247. Springer-Verlag, Berlin, 1987.
8. H.J. Genrich and K. Lautenbach. The Analysis of Distributed Systems by means of Predicate/Transition-Nets. In G. Kahn, editor, *Semantics of Concurrent Compilation*, volume 70 of *Lecture Notes in Computer Science*, pages 123–146. Springer-Verlag, Berlin, 1979.
9. Object Management Group. *OMG Unified Modeling Language 2.5*. OMG, 2013.
10. T. Halpin and T. Morgan. *Information Modeling and Relational Databases*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2008.
11. K.M. van Hee. *Information System Engineering: A Formal Approach*. Cambridge University Press, 1994.
12. R. Hull et al. Business Artifacts with Guard-Stage-Milestone Lifecycles: Managing Artifact Interactions with Conditions and Events. In *International Conference on Distributed Event-Based Systems (DEBS 2011)*. ACM, 2011.
13. K. Jensen. Coloured Petri Nets. In W. Brauer, W. Reisig, and G. Rozenberg, editors, *Advances in Petri Nets 1986 Part I: Petri Nets, Central Models and Their Properties*, volume 254 of *Lecture Notes in Computer Science*, pages 248–299. Springer-Verlag, Berlin, 1987.
14. K. Jensen. *Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use*. Springer-Verlag, Berlin, 1996.

15. N. Lohmann. Compliance by Design for Artifact-Centric Business Processes. In *Business Process Management (BPM 2011)*, volume 6896 of *LNCS*. Springer, 2011.
16. X. Lu, M. Nagelkerke, D. van de Wiel, and D. Fahland. Discovering Interacting Artifacts from ERP Systems. *IEEE Transactions on Services Computing*, 8(6):861–873, 2015.
17. M. de Leoni and W.M.P. van der Aalst. Data-aware Process Mining: Discovering Decisions in Processes Using Alignments. In *Proceedings of the 28th annual ACM symposium on applied computing*, pages 1454–1461. ACM, 2013.
18. A. Nigam and N.S. Caswell. Business Artifacts: An Approach to Operational Specification. *IBM Systems Journal*, 42(3), 2003.
19. E.H. Nooijen, B.F. van Dongen, and D. Fahland. Automatic Discovery of Data-Centric and Artifact-Centric Processes. In *International Conference on Business Process Management*, pages 316–327. Springer, 2012.
20. V. Popova, D. Fahland, and M. Dumas. Artifact Lifecycle Discovery. *International Journal of Cooperative Information Systems*, 24(01):1–44, 2015.
21. A. Rozinat and W.M.P. van der Aalst. Decision Mining in ProM. In *International Conference on Business Process Management*, pages 420–425. Springer, 2006.
22. P.A.C. Verkoulen. *Integrated Information Systems Design: An Approach Based on Object-Oriented Concepts and Petri Nets*. PhD thesis, Eindhoven University of Technology, Eindhoven, 1993.
23. C.R. Zervos. *Coloured Petri Nets: Their Properties and Applications*. PhD thesis, University of Michigan, Michigan, 1977.