Object-Centric Behavioral Constraint Models: A Hybrid Model for Behavioral and Data Perspectives

Guangming Li
Eindhoven University of Technology
Eindhoven, The Netherlands
g.li.3@tue.nl

Renata Medeiros de Carvalho
Eindhoven University of Technology
Eindhoven, The Netherlands
r.carvalho@tue.nl

Wil M.P. van der Aalst
RWTH Aachen University
Aachen, Germany
wvdaalst@pads.rwth-aachen.de

ABSTRACT

In order to maintain a competitive edge, enterprises are driven to improve efficiency by modeling their business processes. Existing process modeling languages often only describe the lifecycles of individual process instances in isolation. Although process models (e.g., BPMN and Data-aware Petri nets) may include data elements, explicit connections to *real* data models (e.g., a UML class model) are rarely made. Therefore, the *Object-Centric Behavioral Constraint (OCBC)* modeling language was proposed to describe the behavioral and data perspectives, and the interplay between them in one single, hybrid diagram. In this paper, we describe OCBC models and introduce the extended interactions between the data and behavioral perspectives on the attribute level. We implement the approach in a plugin and evaluate it by a comparison with other models.

CCS CONCEPTS

• **Applied computing** → **Enterprise computing**; **Business process management**; **Business process modeling**;

KEYWORDS

Business process modeling, Object-Centric, Databases, Class models, Object models, Declarative constraints

ACM Reference format:
Guangming Li, Renata Medeiros de Carvalho, and Wil M.P. van der Aalst. 2019. Object-Centric Behavioral Constraint Models: A Hybrid Model for Behavioral and Data Perspectives. In *Proceedings of ACM SAC Conference, Limassol, Cyprus, April 8-12, 2019 (SAC'19),* 9 pages.
https://doi.org/10.1145/3297280.3297287

1 INTRODUCTION

Information systems are widely used by enterprises and organizations to support their business process executions. Currently, the information systems one encounters in most organizations are *object-centric*, i.e., transactions related to some type of objects (e.g., customer orders) are stored in the same database table (e.g., the "order" table) on data perspective, and events are recorded implicitly (e.g., in redo logs) and separately without a common case id. Some examples of these systems are Customer Relationship Management (CRM) and/or Enterprise Resource Planning (ERP) which provide business functions such as procurement, production, sales, delivery, finance, etc. In this paper, we abstract business processes on such systems as three perspectives: control-flow (i.e., the behavioral perspective), data schema (i.e., the data perspective) and communications (indicating how activities in control-flow impact the corresponding tables in databases) in between, as shown in Figure 1.

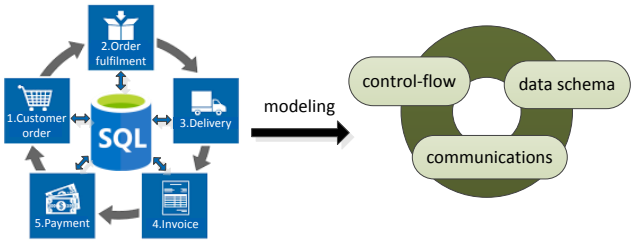

Figure 1: A business process on object-centric information systems can be modeled as three perspectives: control-flow, data schema and communications.

Existing process modeling languages (BPMN diagrams [11], Petri nets [27], EPCs) assume a case notion in business processes, which is used to correlate events for a specific process instance. They work well on process-centric systems such as WFM/BPM systems, but have problems to describe object-centric systems which do not assume a case notion. Besides, they focus on the behavioral perspective, although some models have data elements. For instance, in BPMN diagrams and Data-aware Petri nets (DPNs) [13], activities can write/read data elements (variables and objects), which describe the data perspective and the interactions. However, the more powerful notations used in data modeling language (e.g., a UML class model [8]) are rarely employed. Artifact-centric approaches [7, 12, 23] are state-of-the-art to solve these problems. However, they still force one to pick an instance notion for each artifact, although a case notion for the whole process is not required. Moreover, they can indeed model the data perspective but the control-flow cannot be related to an overall data model (i.e., there is no explicit data model or it is separated from the control-flow) and interactions between different entities are not visible (because artifacts are distributed over multiple diagrams) [16].

The proposed *Object-Centric Behavioral Constraint* (OCBC) modeling language [2, 17] combines ideas from declarative, constraint-based languages like *Declare* [3], and from data/object modeling techniques (ER, UML, or ORM), resulting in OCBC models as shown

SAC'19, April 8-12, 2019, Limassol, Cyprus

ACM ISBN 978-1-4503-5933-7/19/04…$15.00
https://doi.org/10.1145/3297280.3297287

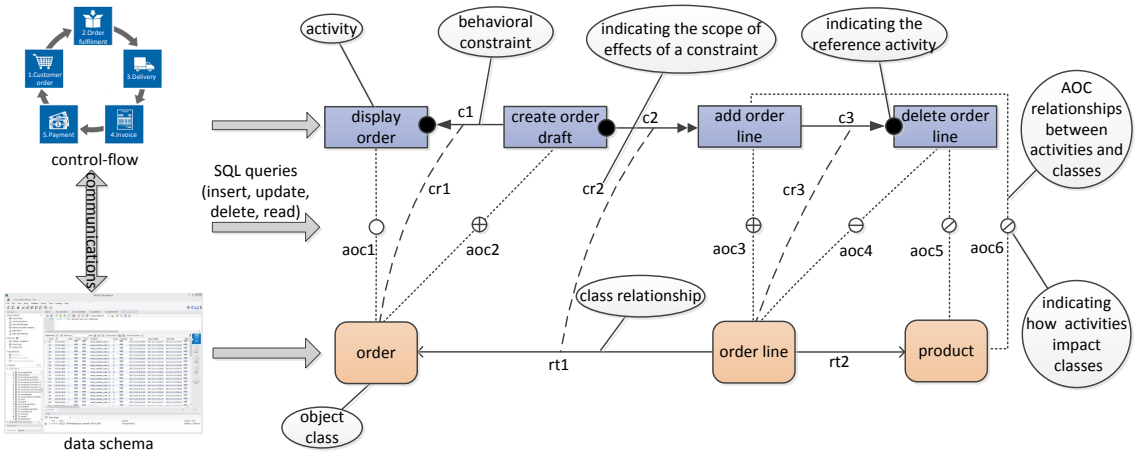

Figure 2: An OCBC model to describe the "order" module of the real information system Dolibarr.

in Figure 2. This paper extends the OCBC modeling language, allowing the definition of attributes for entities (i.e., classes and activities). The communications perspective is strengthened with the annotation of an interaction type (i.e., add, update, delete or read) and the possibility to describe interactions on the attribute level. Besides, a model editor is available to support OCBC models to design business processes.

The remainder is organized as follows. Section 2 briefly introduces OCBC models based on a running example. Section 3 employs class models to describe the data schema. Activity models are defined in Section 4 to describe the restrictions between activities through declarative constraints. Section 5 illustrates OCBC models to combine the above two models. A plugin for designing OCBC models is introduced in Section 6 and Section 8 concludes the paper.

2 RUNNING EXAMPLE

Figure 2 shows an OCBC model example, which describes the business process of the "order" module in the real information system Dolibarr.[1] The model consists of a class model (three classes *order*, *order line* and *product*, and two class relationships $rt1$ and $rt2$) at the bottom, an activity model (four activities *display order*, *create order draft*, *add order line* and *delete order line*, and three constraints $c1$, $c2$ and $c3$) at the top, and six interaction relationships (i.e., $aoc1$,...,$aoc6$) in between. More precisely, the class model describes the data schema of a database with three related tables, e.g., the class "order" corresponds to an "order" table and $rt1$ indicates the "order line" table refers to the "order" table. Four activities and the constraints indicate the control-flow of the business process. For instance, $c1$ means that only after executing the "create order draft" activity, the "display order" activity can be executed (cf. Section 4).

Whenever an activity is executed, an event (an instance of the activity) occurs. It may add (⊕), update (⊘), delete (⊖) or read (○) records in databases. The forms of interactions (indicated by the different "circle" symbols) show how events impact databases. For instance, $aoc2$ means one "create order draft" event adds one record in the "order" table. Note that activities and classes have attributes and interactions can happen on the attribute level by defining a mapping between attributes (cf. Section 5).

Note that, in [2, 17] OCBC models include cardinality constraints on class relationships and interaction relationships. In this paper, we focus on the attribute level of OCBC models, and hide the cardinality constraints for simplicity.

3 MODELING THE DATA PERSPECTIVE

Databases serve as the backbone of object-centric information systems by storing all transactions happened in the business processes supported by these systems. Therefore, the structure of databases should be consistent with the corresponding business processes. As mentioned, object-centric information systems store transactions of the same type in the same database table, e.g., all the orders are stored in the "order" table, as shown in Figure 3. In this paper, we use *classes* to represent tables and *class relationships* to represent the dependency relationships between tables. Classes have attributes as tables have columns, and class relationships connect attributes of classes like that dependency relationships connect foreign keys and primary keys of tables. By integrating the elements mentioned above, we define a *class model* to describe the structure of databases, referring to the notations in [26].

Definition 3.1 (Class Model). Let $\mathcal{U}_{OC}$ be the universe of object *classes*, $\mathcal{U}_{RT}$ be the universe of classes *relationship types*, $\mathcal{U}_{Attr}$ be the universe of *attribute names* and $\mathcal{U}_{Val}$ be the universe of *attribute values*. A class model is a tuple $ClaM = (OC, Attr, RT, key, addi, val, rel)$ such that

- $OC \subseteq \mathcal{U}_{OC}$ is a set of object classes,
- $Attr \subseteq \mathcal{U}_{Attr}$ is a set of attribute names,
- $RT \subseteq \mathcal{U}_{RT}$ is a set of class relationship types,
- $key \in OC \rightarrow \mathcal{P}(Attr) \setminus \{\emptyset\}$ maps each class onto a set of key attribute names,[2]
- $addi \in OC \rightarrow \mathcal{P}(Attr)$ maps each class onto a set of additional attribute names $(key(oc) \cap addi(oc) = \emptyset$ for any class

[1] Dolibarr ERP/CRM is an open source (webpage-based) software package (www.dolibarr.org).

[2] $\mathcal{P}(X)$ is the power set of X, i.e., $Y \in \mathcal{P}(X)$ if $Y \subseteq X$.

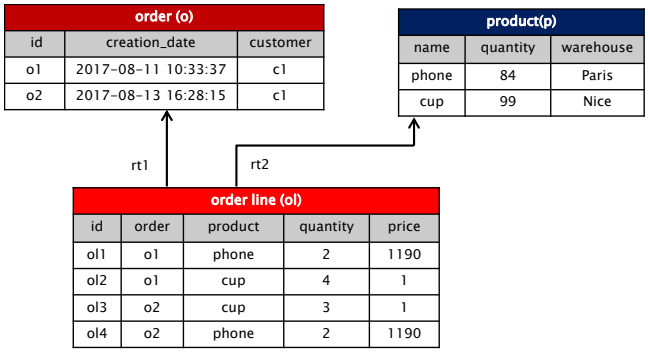

Figure 3: A database with three tables.

$oc \in OC$). Let $Attr_{oc} = key(oc) \cup addi(oc)$ be a shorthand to obtain the key and additional attributes of a class.

- $val \in Attr \to \mathcal{P}(\mathcal{U}_{Val})$ maps each attribute onto a set of possible values,
- $rel \in RT \to (OC \times OC)$ specifies the two classes involved in a relation. Let $rel(rt) = (c_1, c_2)$ for relationship $rt \in RT$: $rel_1(rt) = c_1$ and $rel_2(rt) = c_2$ are shorthand forms to obtain the two individual classes involved in the relationship.

$\mathcal{U}_{ClaM}$ is the universe of class models.

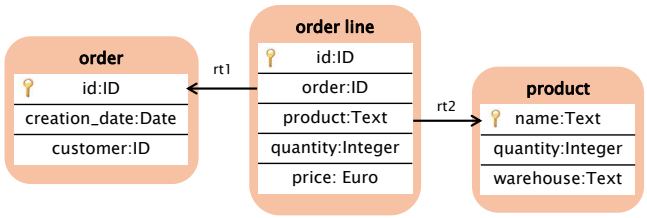

Figure 4: A class model with attributes

Figure 4 shows a class model which describes the database in Figure 3. More precisely, $OC = \{order, product, order\ line\}$ indicates all the tables in the database while $RT = \{rt1, rt2\}$ contains all the dependency relationships between tables. Each class has key attributes indicated by *key* and additional attributes indicated by *addi*, e.g, $key(product) = \{name\}$ and $addi(product) = \{quantity, warehouse\}$.

A class model abstracts a database in a conceptual way. In other words, it defines a space of possible objects. In this paper, we use the term *object* to abstract a record in database tables. An object can be considered as an instance of a class, instantiating all attributes of the class. For example, a record $o1$ in the "order" table (the first row) in Figure 3 can be considered as an object of class "order". Each value (e.g., "c1") in the record can be considered as an attribute of the object.

Definition 3.2 (Object Model). Let $ClaM = (OC, Attr, RT, key, addi, val, rel)$ be a class model and $M^{ClaM} = \{map \in Attr \nrightarrow \mathcal{U}_{Val} \mid \forall attr \in dom(map) : map(attr) \in val(attr)\}$ be the set of mappings of $ClaM$. $O^{ClaM} = \{(oc, map) \in OC \times M^{ClaM} \mid dom(map) = Attr_{oc}\}$ is the set of all objects of $ClaM$. $O_{null} = \{(oc, map_{null}) \mid oc \in OC \wedge map_{null} \in \emptyset \to \mathcal{U}_{Val}\}$ is the set of inexistent (uninitialized) objects. An object model of $ClaM$ is a set of objects $OM \subseteq O^{ClaM}$. Let VOM be the set of valid object models of $ClaM$. $OM \in VOM$ if and only if $\forall (oc, map), (oc, map') \in O^{ClaM} : ((\forall attr \in key(oc) : map(attr) = map'(attr)) \Rightarrow map = map')$, i.e., key values must be unique.

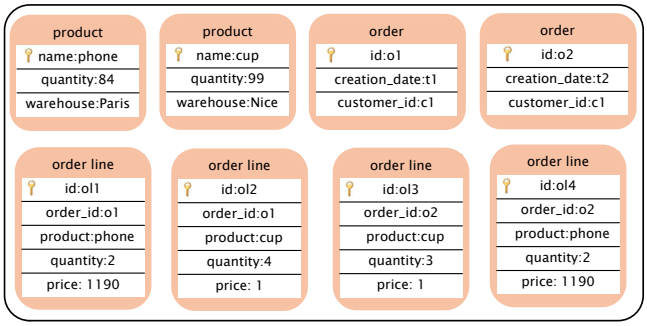

Figure 5: An object model with attributes.

An object (oc, map) assigns a proper value to each attribute of its corresponding class oc, indicated by map. For instance, the first record in the "product" table in Figure 3 can be formalized as (oc, map) where $oc = product$, $dom(map) = \{name, quantity, warehouse\}$, and $map(name) = phone$, $map(quantity) = 84$ and $map(warehouse) = Paris$. An object model is a set of objects and it is valid if each object has a unique id, i.e., there do not exist two objects which have the same value for each key attribute. An object model represents the state of a database at some moment. For instance, Figure 5 shows an object model representing the state of the database in Figure 3. This object model contains 8 objects, i.e., $OM = \{o1, o2, ol1, ol2, ol3, ol4, phone, cup\}$.

4 MODELING THE BEHAVIORAL PERSPECTIVE

The behavioral perspective of a business process indicates the constraints between activities. There exist temporal restrictions on activities, e.g., "display order" activity can happen only after "create order draft" activity happens. In this section, we abstract the restrictions as behavioral constraints between activities. For instance, the restriction mentioned above can be described as a constraint between activity "create order draft" and activity "display order".

In object-centric information systems, the behavioral perspective is quite flexible. It should be generic enough to enable users to deal with the flexible and dynamic business environments. Therefore, we choose declarative constraints inspired by *Declare* [3] to model restrictions. Compared with procedural languages like Petri nets, declarative languages are "open" languages, i.e., they allow for anything which is not explicitly forbidden by constraints, and they are more suitable for modeling flexible business processes. A constraint has a reference activity, a target activity and a corresponding constraint type which specifies the restriction.

Definition 4.1 (Constraint Types). $\mathcal{U}_{CT} = \{X \subseteq \mathbb{N} \times \mathbb{N} \mid X \neq \emptyset\}$ defines the universe of all possible constraint types. Any element of

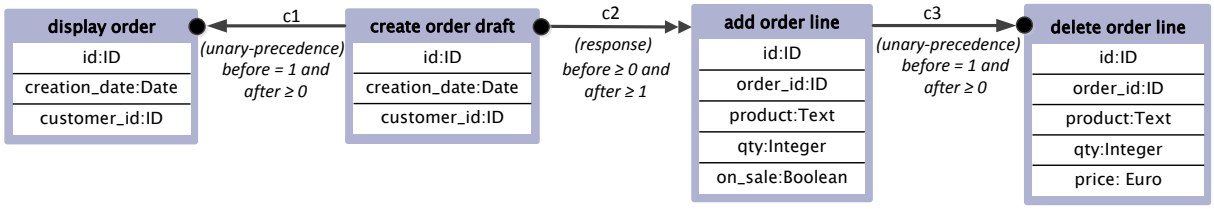

Figure 6: An activity model with attributes.

$\mathcal{U}_{CT}$ 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 6 shows an activity model with three constraints $c1$, $c2$ and $c3$. The black dot indicates the reference activity and the arrow indicates the constraint type. For instance, "display order" is the reference activity and "create order draft" is the target activity of $c1$. Constraints $c1$ and $c3$ correspond to the constraint type $\{(before, after) \in \mathbb{N} \times \mathbb{N} \mid before = 1 \wedge after \geq 0\}$ (referred to as *unary-precedence* and depicted as a single arrow) and constraint $c2$ corresponds to the constraint type $\{(before, after) \in \mathbb{N} \times \mathbb{N} \mid before \geq 0 \wedge after \geq 1\}$ (referred to as *response* and depicted as a double arrow).

Here, we only give two constraint types, but actually all possible constraint types are allowed in activity models. For instance, in order to make the model in Figure 6 more precise, one can add negative constraints, e.g., a *non-response* ($\{(before, after) \in \mathbb{N} \times \mathbb{N} \mid before \geq 0 \wedge after = 0\}$) constraint between "create order draft" (target activity) and "display order" (reference activity), which requires that "create order draft" cannot happen after "display order". [2] shows the graphical representations and semantics for more constraint types. An activity model is formally defined as follows.

Definition 4.2 (Activity Model). Let $\mathcal{U}_A$ be the universe of activities. An *activity model* is a tuple $ActM = (A, Attr', C, \pi_{ref}, \pi_{tar}, type, addi', val')$, where

- $A \subseteq \mathcal{U}_A$ is a set of activities (denoted by rectangles),
- $Attr' \subseteq \mathcal{U}_{Attr}$ is a set of attribute names,
- C is a set of constraints (denoted by various types of edges),
- $\pi_{ref} \in C \rightarrow A$ defines the reference activity of a constraint (denoted by a black dot connecting constraint and activity),
- $\pi_{tar} \in C \rightarrow A$ defines the target activity of a constraint (the side without a black dot), and
- $type \in C \rightarrow \mathcal{U}_{CT}$ specifies the type of each constraint (denoted by the type of edge).
- $addi' \in A \rightarrow \mathcal{P}(Attr)$ maps each activity onto a set of additional attribute names,and
- $val' \in Attr \rightarrow \mathcal{P}(\mathcal{U}_{Val})$ maps each attribute onto a set of possible values.

An activity model is a collection of activities and constraints. Figure 6 shows an activity model $ActM = (A, Attr', C, \pi_{ref}, \pi_{tar}, type, addi', val')$ with four activities, i.e., $A = \{create\ order\ draft, display\ order, add\ order\ line, delete\ order\ line\}$, eight attributes, i.e., $Attr' = \{id, create\_date, \ldots, price\}$, and three constraints, i.e., $C = \{c1, c2, c3\}$. Consider the constraint $c1$ as example. $\pi_{ref}(c1) = display\ order$, $\pi_{tar}(c1) = create\ order\ draft$, $type(c1) = \{(before, after) \in \mathbb{N} \times \mathbb{N} \mid before = 1 \wedge after \geq 0\}$ (unary-precedence). $addi'(display\ order) = \{id, create\_date, customer\}$ and $val'(id) = ID$ (representing $\mathbb{N}$).

The reference (target) activity of a constraint defines the corresponding set of reference (target) events. For each reference event, the number of its related target events should meet the requirements of the constraint. In traditional process modeling notations, a constraint is defined for one process instance (case) in isolation. This means that the related target events for a reference event are all target events corresponding to the same case. As discussed before, the case notion is often too rigid. There may be multiple case notions at the same time, causing one-to-many or many-to-many relations. In OCBC models, events are correlated based on the data perspective using the approaches in [2, 15], which is out of the scope of this paper.

Definition 4.3 (Events). Let $ActM = (A, Attr', C, \pi_{ref}, \pi_{tar}, type, addi', val')$ be an activity model and $M^{ActM} = \{map \in Attr' \nrightarrow \mathcal{U}_{Val} \mid \forall attr \in dom(map) : map(attr) \in val'(attr)\}$ be the set of mappings of $ActM$. $E^{ActM} = \{(a, map) \in A \times M^{ActM} \mid dom(map) = addi'(a)\}$ is the set of all events of $ActM$.

Each event is related to an activity and instantiates the attributes of the activity. An event example is $e = (create\ order\ draft, map)$ where *create order draft* is an activity, and $map(creation\_date) = t1$, $map(id) = o1$ and $map(customer\_id) = c1$ (i.e., the first event in Figure 8).

5 OBJECT-CENTRIC BEHAVIORAL CONSTRAINT MODELS

Up to now, we have introduced class models, i.e., classes and class relationships, and activity models, i.e., activities and constraints. In this section, we combine them through interaction models to cover the three perspectives in Figure 1 in one single diagram.

Definition 5.1 (Object-Centric Behavioral Constraint Model). An *object-centric behavioral constraint model* is a tuple $OCBCM = (ClaM, ActM, InterM)$, where

- $ClaM = (OC, Attr, RT, key, addi, val, rel)$ is a class model,
- $ActM = (A, Attr', C, \pi_{ref}, \pi_{tar}, type, addi', val')$ is an activity model, and
- $InterM = (AOC, form, linkAttr)$ is an interaction model between $ClaM$ and $ActM$ where
  - $AOC \subseteq A \times OC$ is a set of relationships between activities and classes,

– $form \in AOC \rightarrow OP$ gives the type of a relationship, i.e., how the activity impacts the class, where $OP = \{\oplus, \oslash, \ominus, \bigcirc\}$,
– $linkAttr \in AOC \times Attr \nrightarrow Attr'$ maps a pair of a relationship and an attribute from its class onto an attribute from its activity.

For convenience, we say that an event $e = (a, mapE)$ corresponds to an object $o = (oc, map)$ by $aoc = (a, oc)$, denoted as $e \overset{aoc}{\sim} o$ if $\forall attr \in key(oc) : map(attr) = mapE(attr')$ where $attr' = linkAttr((aoc, attr))$.

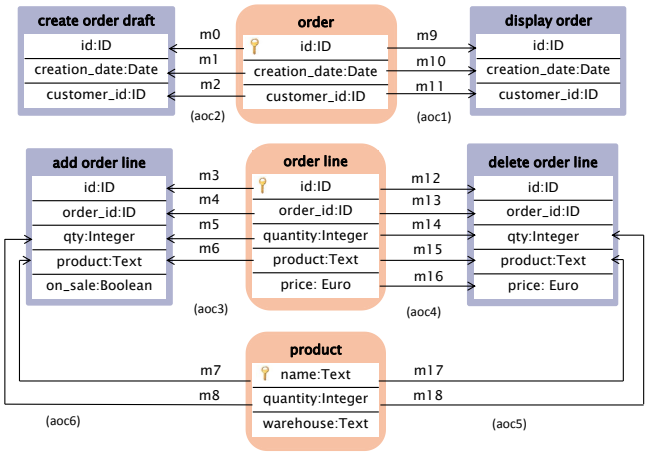

Figure 7: The interactions (corresponding to aoc1,...,aoc6 in Figure 2) on attribute level.

An interaction model *InterM* mainly consists of a set of *AOC relationships* between (A)ctivities and (O)bject (C)lasses, which can be described by two levels. For instance, Figure 2 shows six AOC relationships ($aoc1, \ldots, aoc6$) on the entity level and Figure 7 shows the interactions on the attribute level. Each relationship is represented by a dashed line with a circle in the middle on the entity level, and directed edges between class attributes and activity attributes on the attribute level. Function *form* gives the type of each relationship, i.e., $\oplus$ (insert), $\oslash$ (update), $\ominus$ (delete) and $\bigcirc$ (read). Consider $aoc3 = (add\ order\ line, order\ line)$ in Figure 2 as an example. $form(aoc3) = \oplus$, denoting that one "add order line" event inserts an "order line" object,

Function *linkAttr* builds a mapping between activity attributes and class attributes for each AOC relationship, as shown in Figure 7. For instance, $m3$, $m4$, $m5$ and $m6$ show the interaction on the attribute level for $aoc3$ and $linkAttr(aoc3, quantity) = qty$, indicated by $m5$. Note that, for each relationship (a, oc), *linkAttr* must connect each key attribute of oc to an attribute of a.[3] For instance, the attribute "price" of "order line" is allowed to have no connected activity attribute, but the attribute *id* must be connected to an activity attribute since *id* is the key attribute of "order line". Besides, we assume that *linkAttr* is injective, i.e., it is impossible two class attributes correspond to the same activity attribute.

[3]The domain of *linkAttr* $dom(linkAttr) \supseteq \{((a, oc), attr) \mid (a, oc) \in AOC \wedge attr \in key(oc)\}$.

An OCBC model consists of a class model, an activity model and an interaction model. An example is shown in Figure 2. The formalization of effects of events from the activity model on objects of the class model is given in Definition 5.2. Such interactions (effects) are given in terms of AOC relationships in the interaction model.

Definition 5.2 (Effect of an Event). Let $InterM = (AOC, form, linkAttr)$ be an interaction model. Through an AOC relationship $aoc = (a, oc) \in AOC$, an event $e = (a, mapE)$ changes an object from the old state $o_{old} = (oc, map_{old})$ to the new state $o_{new} = (oc, map_{new})$, denoted as $o_{old} \xrightarrow[aoc]{e} o_{new}$, if and only if

(1) $form(aoc) = \oplus$, then $o_{old} \in O_{null}$, $o_{new} \in O^{ClaM}$ and $\forall attr \in dom(map_{new})$:
- $map_{new}(attr) = mapE(attr')$ if $attr' \in dom(mapE)$, and
- $map_{new}(attr) = NULL$ otherwise,[4]

(2) $form(aoc) = \oslash$, then $e \overset{aoc}{\sim} o_{old}$ and $\forall attr \in dom(map_{new})$:
- $map_{new}(attr) = oper(mapE(attr'), map_{old}(attr))$ if $attr' \in dom(mapE)$, and
- $map_{new}(attr) = map_{old}(attr)$ otherwise,

(3) $form(aoc) = \ominus$, then $e \overset{aoc}{\sim} o_{old}$ and $o_{new} \in O_{null}$, and

(4) $form(aoc) = \bigcirc$, then $e \overset{aoc}{\sim} o_{old}$ and $o_{new} = o_{old}$,

where $attr' = linkAttr((aoc, attr))$ and *oper* is a given function based on domain knowledge (explained next).

Definition 5.2 employs some rules to describe how an event impacts objects through one AOC relationship. More precisely, rule 1 indicates that an "add" event (i.e., the type of the AOC relationship is $\oplus$) creates a new object. Each attribute of the object is initialized as the same value as its corresponding event attribute (indicated by function *linkAttr*), or "NULL" if it has no corresponding event attribute. Rule 2 means that an event updates its corresponding object (which is identified by the operator $\sim$, cf. Definition 5.1). More precisely, each attribute of the object is updated by *oper*, or remains unchanged if it has no corresponding event attribute. Note that *oper* gives a way to compute the new value of an object attribute (i.e., $map_{new}(attr)$) based on the corresponding event attribute value (i.e., $mapE(attr')$) and the old value of the object attribute (i.e., $map_{old}(attr)$). This enables complex operations (e.g., summations and subtraction) to update object attributes based on domain knowledge. For instance, we can make $oper(a, b) = a$ if we want that the object value is updated as the event value, or $oper(a, b) = b - a$ if we want that the new object value is the result of subtracting the event value from the old object value. Rule 3 denotes that an event removes its corresponding object while, for rule 4, an event reads its corresponding object.

Figure 8 gives an example to show how events change objects in the object model through AOC relationships in Figure 7. For instance, the first event inserts an "order" object $o1$ through the AOC relationship $aoc2$, copying all its attributes to the object. The second event inserts an "order line" object $ol1$ through $aoc3$ and updates the attribute "quantity" of the "product" object *phone* through $aoc6$. Similarly, the third event adds an "order line" object $ol2$ and updates the "product" object *cup*. The fourth event reads the information of the "order" object $o1$ without changing anything. The last

[4]*NULL* means that the attribute *attr* is blank after the creation of its corresponding object.

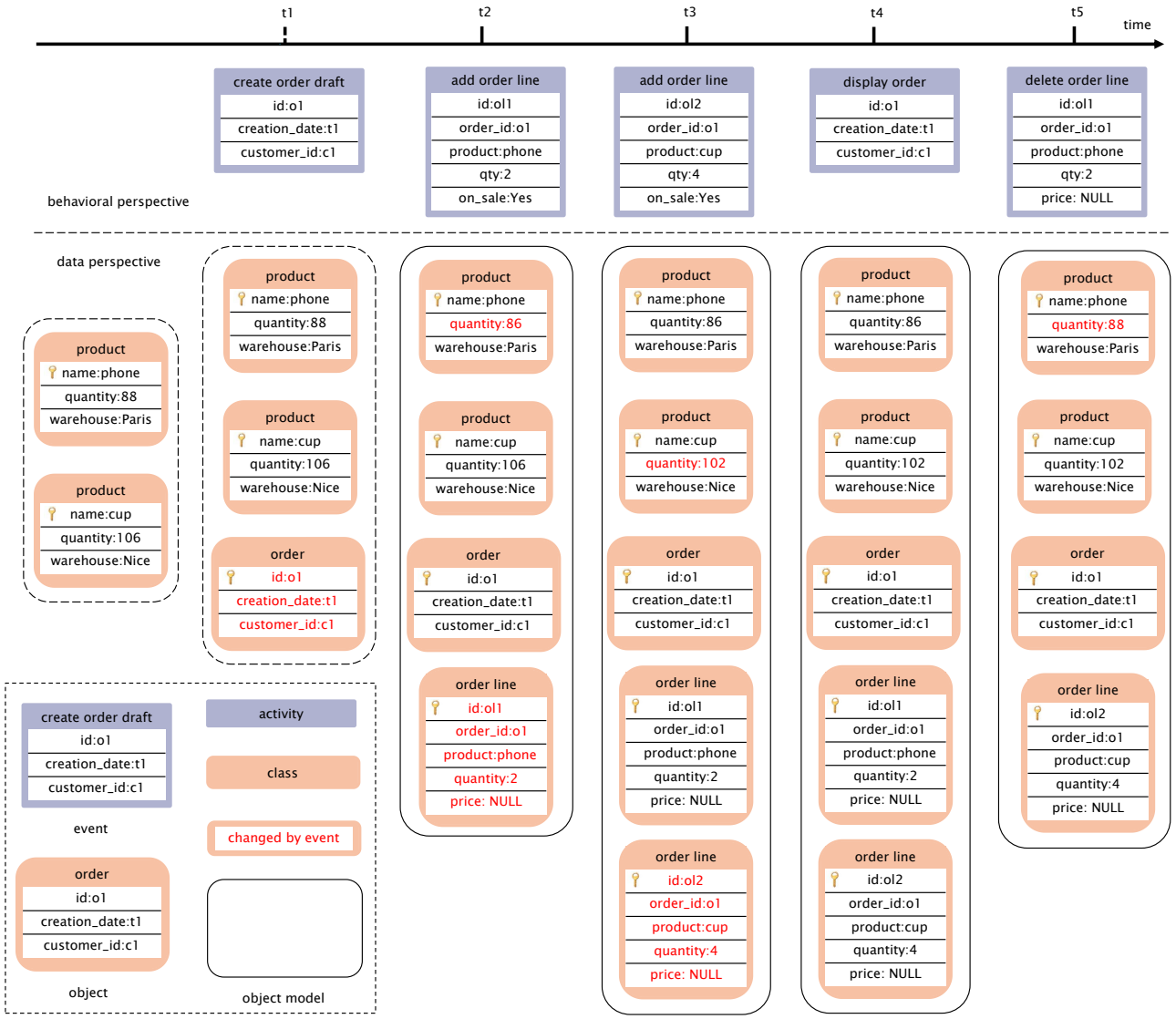

Figure 8: An event may add, update, delete or read one or more objects in the object model.

event removes the "order line" object $ol1$ through $aoc4$ and updates the "product" object *phone* through $aoc5$. Note that an "add order line" event updates the attribute "quantity" of an "product" object based on $oper(a, b) = b - a$, i.e., subtracting the value of the event attribute "qty" from the old value of the object attribute "quantity". For instance, at the moment $t2$, "quantity" of the object *phone* is updated as "86", since its old value is "88" and the value of the event attribute "qty" is "2". In contrast, a "delete order line" event updates the attribute "quantity" based on $oper(a, b) = b + a$. For instance, at the moment $t5$, "quantity" of the object *phone* is updated as "88", since its old value is "86" and the value of the event attribute "qty" is "2". One event can change multiple objects. In the database context, such an event corresponds to a transaction consisting of several SQL statements. A database system must ensure proper execution of transactions despite failures, i.e., either the entire transaction executes, or none of it does.

6 EVALUATION

We have implemented the OCBC modeling language as a plugin "OCBC Model Editor" in ProM nightly builds (installing the *OCBC package*) to support designing business processes.[5] Figure 9 shows the interface of the plugin and a designed model in panel ⑤. By dragging a node from panel ② to panel ⑤, one can add an activity or a class. When locating the mouse cursor in the center of a node, one can drag out an edge for the node to connect another node. When a node or an edge is selected, it is highlighted in red and

[5] Access *http://www.win.tue.nl/ocbc/* for more information, such as tools, manuals and OCBC model examples.

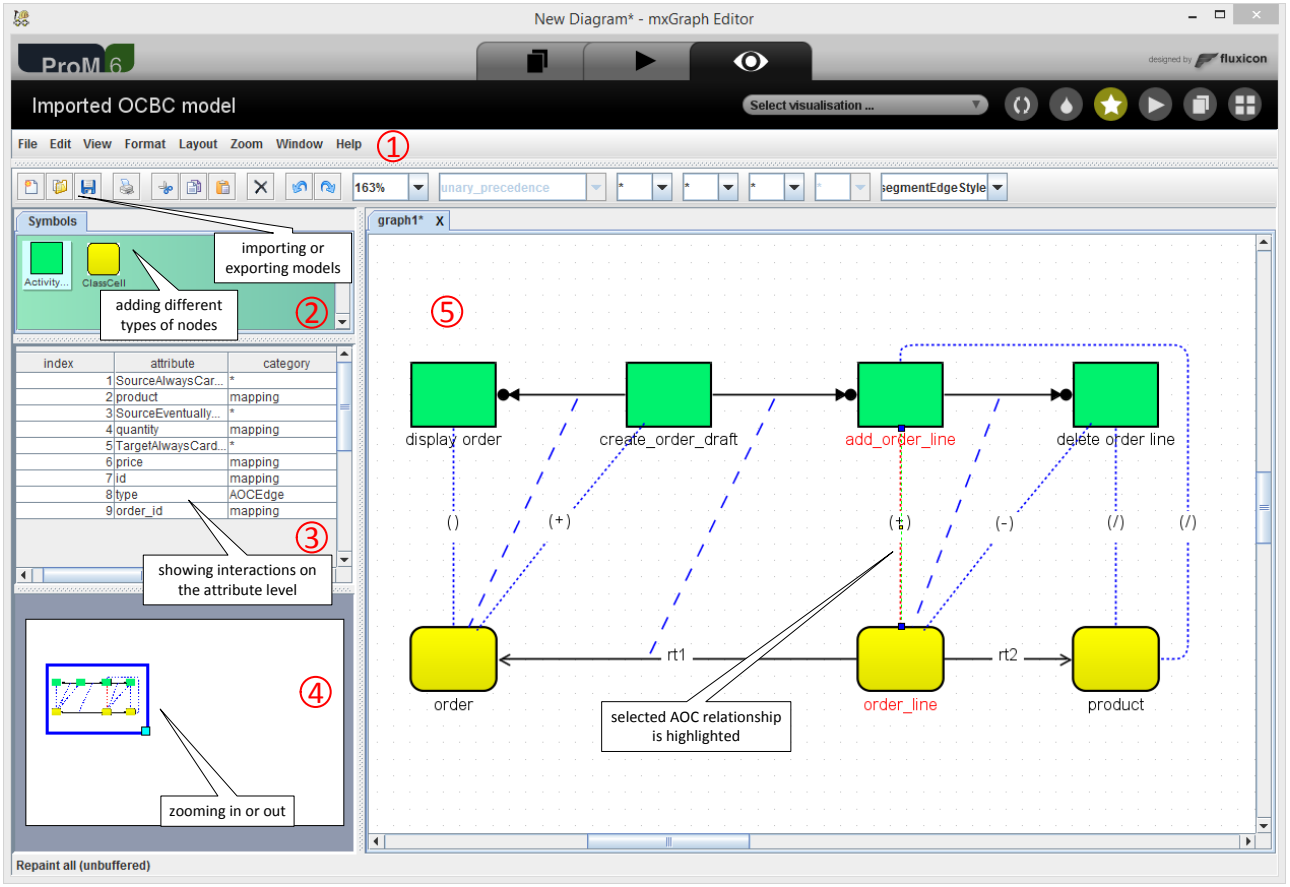

Figure 9: The OCBC model editor in ProM.

its attribute information can be viewed or edited in panel ③. It is possible to zoom in/out models by operating panel ④. After designing a model, one can export the model in panel ①.

The idea of evaluating OCBC models is to compare its ability with other models by describing the same business process. The OTC (order-to-cash) business process is a typical and significant scenario in enterprises. It is supported by ERP systems, e.g., Dolibarr, an open source ERP system for small and medium enterprises. The OTC business process covers a range of modules from creating orders to paying bills. Figure 10 employs an informal notation to describe the business process of "order" module (including the behavioral perspective, data perspective and the communications in between) in Dolibarr.

First, we explain the behavioral (i.e., control-flow) perspective including four activities and three constraints. The edges with single arrows indicate the temporal orders between activities while the cardinalities on edges indicate the restriction on the correspondences (e.g., one-to-many) between activities. For instance, the edge ② means that after a "create order draft" event, one or more "add order line" events can happen. Indicated by ③, each "add order line" event is followed by at most one "delete order line" event, and each "delete order line" event must have precisely one corresponding "add order line" event before.[6]

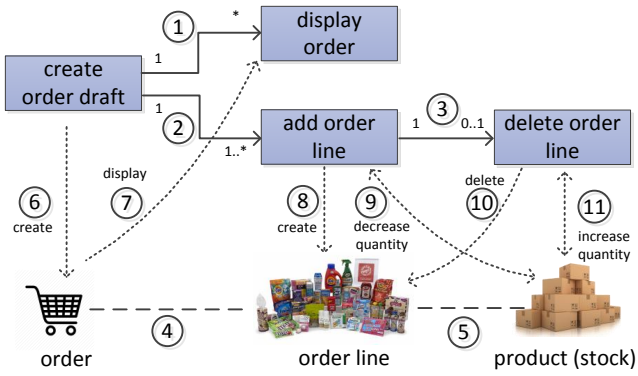

Figure 10: An informal model to describe the "order" module in the OTC business process in Dolibarr.

[6] An event a having a corresponding event b before does not mean that a happens directly after b, i.e., it is possible that other events happen between b and a.

Events on the behavioral perspective may add, update or delete objects on the data perspective. The process in Figure 10 contains three types of objects, i.e., "order", "order line" and "product", and ④ and ⑤ indicate the associations between these types. Dotted lines with arrows are employed to show the interactions between the behavioral perspective and data perspective. For instance, ⑥ denotes that a "create order draft" event creates an "order" object and ⑦ means that a "display order" event displays all contents in an order. ⑧ and ⑨ indicate that an "add order line" event adds an order line to the order and meanwhile decreases the quantity of the corresponding product, respectively. In contrast, ⑩ and ⑪ signify that an "add order line" event removes an order line from the order and meanwhile increases the quantity of the corresponding product, respectively.

Figure 2 employs an OCBC model to describe the process explained above. Note that the process is quite flexible and there exist one-to-many relations (i.e., multiple instances), e.g., one "create order draft" event may correspond to multiple "add order line" events. In the OCBC model, we do not assume a case notion, and use the data perspective to correlate events and distinguish multiple instances. Based on this, declarative constraints are used to describe the process more precisely. The interaction types indicate how events on the behavioral perspective impact (i.e., add, update, delete or read) the objects on the data perspective. Besides, Figure 4, Figure 6 and Figure 7 show the attribute level of the process from different perspectives. By doing this, the interactions are presented more clearly. For instance, with the support of interaction types and the attribute level, one can know that an "add order line" event updates the "quantity" value of a "product" object .

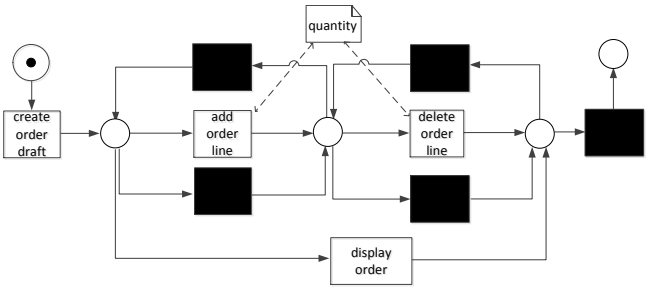

Figure 11: A DPN example to describe the "order" module.

Next, we use other modeling techniques to describe the process. Figure 11 presents the process with a DPN diagram. Due to the one-to-many relation (② in Figure 10), "add order line" and "delete order line" events can happen multiple times in the process. Accordingly, the DPN diagram employs several silent transitions (and loops) to allow the occurrence of multiple instances, resulting in an underfitting model (allowing to much behavior). For instance, the constraint (③ in Figure 10) indicating that a "delete order line" event can happen only after its corresponding "add order line" event is not contained in the model. DPN diagrams support modeling the data perspective with variables. For instance, "quantity" is a variable which indicates that "add order line" and "delete order line" events modify the quantity of corresponding product.

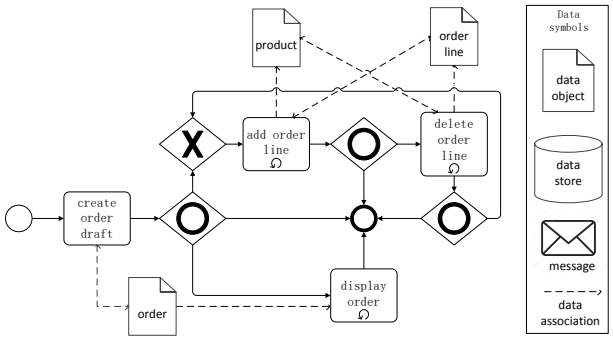

Figure 12: A BPMN example to describe the "order" module.

Figure 12 employs a BPMN diagram to describe the "order" module. Its behavioral perspective is also too general, and fails to present the constraint between activities "delete order line" and "add order line". The BPMN notation has data symbols to support modeling the data perspective of business processes, e.g., data object, data store, message and data association. In Figure 12, three data objects are created to show the interactions between the behavioral perspective and data perspective. However, the objects are considered in isolation (there are no relations between objects) and the interactions on the attribute level are not shown, e.g., "add order line" events modify the "quantity" attribute of the product object.

In summary, in the flexible scenario with multiple instances, DPN and BPMN notations suffer an underfitting behavioral perspective. Furthermore, the data perspective and interactions between events and data in both models are not as powerful as those in the OCBC model. In contrast, OCBC models can explicitly represent one-to-many and many-to-many relations, as well as interactions between different process instances in a single, hybrid diagram.

7 RELATED WORK

Process modeling languages play an important role in the design and analysis of business processes. Over time researchers have proposed various languages and models to describe business processes [10, 21, 24].

Most models mainly focus on one aspect of business processes. For instance, Petri nets [27] and Declare models [3] are both focusing on the behavioral perspective to describe the workflows. In contrast, Petri nets are procedural and strict while Declare models are declarative and flexible. ER diagrams mainly describe the data perspective, e.g., the structure of the databases corresponding to business processes. Besides, Integration DEFinition (IDEF) [20], Unified Modeling Language (UML) [5, 8] and Systems Modeling Language (SysML) [4, 9] provide a bunch of diagrams, in which each diagram mainly models processes from a specific angle.

There exist some models which can describe multiple perspectives in one single diagram. Consider for example the various types of colored Petri nets, i.e., Petri nets add "color" on tokens to represent attributes and values [13]. Data-aware Petri nets (DPNs) [19, 22] model the data elements by writing/reading variables and adding guards on transitions. BPMN [11], Data flow chart [6, 14] and UML activity diagram can describe behavioral perspective

and its communication with data perspective by data objects (e.g., documents) and data stores (e.g., tables). [25] provides a concrete framework, called RAW-SYS, to model the control-flow, the data and their interaction, and to verify data-aware processes. Artifact-centric approaches [7, 12, 23] (including the earlier work on proclets [1]) aim to capture business processes in terms of so-called business artifacts, i.e., key entities which drive a company's operations and whose lifecycles and interactions define an overall business process. Artifacts have data and lifecycles attached to them, thus relating both perspectives.

These process-centric approaches (e.g., DPN and BPMN diagrams) can describe the data perspective to some extent. They try to consider the data perspective by adding data elements, such as attributes, values, variables and objects, onto the control-flow. However they do not support explicit data modeling as can be found in ER models. Artifact-centric approaches need to identify artifacts beforehand based on domain knowledge, and within an artifact (proclet, or subprocess), one is forced to pick a single instance notion. Besides, the description of the end-to-end behavior needs to be distributed over multiple diagrams (e.g., one process model per artifact).

Compared with existing approaches, an OCBC model can describe the data perspective, the behavioral perspective and the interplay in between in one single diagram, which provides a picture of the whole system. Besides, it is powerful on presenting the interactions through mapping class attributes onto activity attributes.

8 CONCLUSION AND FUTURE WORK

An OCBC model is a hybrid model consisting of: a class model describing the structure of databases; an activity model describing the possible events and restrictions between events; and an interaction model describing how events modify the database tables. This paper strengthens the proposed OCBC models by (i) adding attribute onto classes and activities, (ii) adding interaction types on AOC relationships and (iii) formalizing the interactions (i.e., how events change objects) on the attribute level.

OCBC models outperform existing approaches when describing flexible business processes on object-centric information systems, since it is more powerful to model communications and shows three perspectives of a business process in one single diagram. Besides, we implemented the language as an editor, which supports designing models to describe business processes.

Moreover, the OCBC models serve as a starting point for a new line of research. Based on event logs extracted from data generated by object-centric information systems [18], automatically discovering OCBC models [17] should be further developed to incorporate the discovery of attributes and the interactions related. The conformance checking techniques [2] will be able to detect different kinds of deviations taking the attribute level into consideration. Finally, the behavioral perspective can be extended to incorporate data-aware constraints (which may be comparable to the guards in DPN diagrams).

REFERENCES

[1] W.M.P. van der Aalst, P. Barthelmess, C.A. Ellis, and J. Wainer. 2001. Proclets: A Framework for Lightweight Interacting Workflow Processes. *International Journal of Cooperative Information Systems* 10, 4 (2001), 443–481.

[2] W.M.P. van der Aalst, G. Li, and M. Montali. 2017. *Object-Centric Behavioral Constraints.* CoRR Technical Report. arXiv.org e-Print archive. Available at https://arxiv.org/abs/1703.05740.

[3] W.M.P. van der Aalst, M. Pesic, and H. Schonenberg. 2009. Declarative Workflows: Balancing Between Flexibility and Support. *Computer Science - Research and Development* 23, 2 (2009), 99–113.

[4] L. Balmelli et al. 2007. An Overview of the Systems Modeling Language for Products and Systems Development. *Journal of Object Technology* 6, 6 (2007), 149–177.

[5] G. Booch. 2005. *The Unified Modeling Language User Guide.* Pearson Education India.

[6] P.D. Bruza and T. van der Weide. 1989. *The Semantics of Data Flow Diagrams.* Citeseer.

[7] D. Cohn and R. Hull. 2009. Business Artifacts: A Data-Centric Approach to Modeling Business Operations and Processes. *IEEE Data Engineering Bulletin* 32, 3 (2009), 3–9.

[8] H.E. Eriksson and M. Penker. 2000. Business Modeling with UML. *New York* (2000), 1–12.

[9] S. Friedenthal, A. Moore, and R. Steiner. 2014. *A Practical Guide to SysML: The Systems Modeling Language.* Morgan Kaufmann.

[10] G.M. Giaglis. 2001. A Taxonomy of Business Process Modeling and Information Systems Modeling Techniques. *International Journal of Flexible Manufacturing Systems* 13, 2 (2001), 209–228.

[11] Object Management Group. 2010. *Business Process Model and Notation.* OMG.

[12] R. Hull et al. 2011. 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.

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

[14] P.G. Larsen, N. Plat, and H. Toetenel. 1994. A Formal Semantics of Data Flow Diagrams. *Formal aspects of Computing* 6, 6 (1994), 586–606.

[15] G. Li, R.M.de Carvalho, and W.M.P. van der Aalst. 2018. Configurable Event Correlation for Process Discovery from Object-Centric Event Data. In *2018 IEEE International Conference on Web Services (ICWS).* 203–210. https://doi.org/10.1109/ICWS.2018.00033

[16] G. Li and R.M. de Carvalho. 2018. Dealing with Artifact-Centric Systems: a Process Mining Approach. In *Proceedings of the 9th EMISA, May 24–25, 2018.* 80–84.

[17] G. Li, R.M. de Carvalho, and W.M.P. van der Aalst. 2017. Automatic Discovery of Object-Centric Behavioral Constraint Models. In *Business Information Systems: 20th International Conference, BIS 2017, June 28–30, 2017, Proceedings.* Springer, 43–58.

[18] G. Li, E. González López de Murillas, R.M. de Carvalho, and W.M.P. van der Aalst. 2018. Extracting Object-Centric Event Logs to Support Process Mining on Databases. In *Information Systems in the Big Data Era, CAiSE Forum 2018,* Jan Mendling and Haralambos Mouratidis (Eds.). Springer International Publishing, Cham, 182–199.

[19] M. de Leoni and W.M.P. van der Aalst. 2013. Data-aware Process Mining: Discovering Decisions in Processes Using Alignments. In *Proceedings of the 28th annual ACM symposium on applied computing.* ACM, 1454–1461.

[20] C. Menzel and R.J. Mayer. 1998. The IDEF Family of Languages. In *Handbook on architectures of information systems.* Springer, 209–241.

[21] H. Mili et al. 2010. Business Process Modeling Languages: Sorting Through the Alphabet Soup. *ACM Computing Surveys (CSUR)* 43, 1 (2010), 4.

[22] N. Sidorova, C. Stahl and N. Trčka. 2011. Soundness verification for conceptual workflow nets with data: Early detection of errors with the most precision possible. *Information Systems* 36, 7 (2011), 1026–1043.

[23] A. Nigam and N.S. Caswell. 2003. Business Artifacts: An Approach to Operational Specification. *IBM Systems Journal* 42, 3 (2003), 428–445.

[24] O.S. Noran. 2000. Business Modelling: UML vs. IDEF. *School of Computing and Information Technology, Griffith University* (2000).

[25] R. De Masellis, C. Di Francescomarino, C. Ghidini, M. Montali and S. Tessaris. 2017. Add Data into Business Process Verification: Bridging the Gap between Theory and Practice.. In *AAAI.* 1091–1099.

[26] W.M.P. van der Aalst. 2015. Extracting Event Data from Databases to Unleash Process Mining. In *BPM-Driving innovation in a digital world.* Springer, 105–128.

[27] W.M.P. van der Aalst and K.M. van Hee. 1996. Business Process Redesign: A Petri-Net-Based Approach. *Computers in industry* 29, 1-2 (1996), 15–26.