Modeling Business Processes Using Object-Centric Behavioral Constraints

Guangming Li
Eindhoven University of Technology, P.O. Box 513, 5600 MB, Eindhoven, The Netherlands.

g.li.3@tue.nl

Abstract. Today’s process modeling languages often force the analyst or modeler to straightjacket real-life processes into simplistic or incomplete models that fail to capture the essential features of the domain under study. Conventional business process models only describe the lifecycles of individual instances (cases) in isolation. Although process models may include data elements (cf. BPMN), explicit connections to real data models (e.g., an entity relationship model or a UML class model) are rarely made. Therefore, we propose a novel notation that extends data models with a behavioral perspective. Data models can easily deal with many-to-many and one-to-many relationships. This is exploited to create process models that can also model complex interactions between different types of instances. Classical multiple-instance problems are circumvented by using the data model for event correlation. The declarative nature of the proposed language makes it possible to model behavioral constraints over activities like cardinality constraints in data models. The resulting object-centric behavioral constraint model is able to describe processes involving interacting instances and complex data dependencies. In particular, we illustrate the usefulness of this new approach by a motivating example from the ERP/CRM system Dolibarr.

1 Introduction

Process mining represents a set of techniques which are widely used to discover process models, and check conformance based on these models. In these techniques, process modeling languages (or process presentation notations) decide the presentational bias of models and have a significant impact on latter analysis such conformance checking. Today’s process modeling languages often focus on analyzing process-centric systems, in which an explicit case notion exists, to describe the lifecycles of individual instances (cases) in isolation, with ignoring or underestimating the data perspective. However, there is a rising trend that more and more enterprises are employing object-centric information systems, such as SAP, ERP and CRM, to deal with their business transactions. Existing process modeling languages (e.g., BPMN diagrams, Workflow nets, EPCs, or UML activity diagrams) tend to suffer from two main problems:
– It is difficult to model interactions between process instances, which are in fact typically considered in isolation. Concepts like lanes, pools, and message flows in conventional languages like BPMN aim to address this. However, within each (sub)process still a single instance is modeled in isolation.
It is also difficult to model the data-perspective and control-flow perspective in a unified and integrated manner. Data objects can be modeled, but the more powerful constructs present in Entity Relationship (ER) models and UML class models cannot be expressed well in process models. For example, cardinality constraints in the data model must influence behavior, but this is not reflected at all in today’s process models.

In order to solve the above problems, a novel modeling language named object-centric behavioral constraint modeling language is proposed, which extends data models with a behavioral perspective. Data models can efficiently describe data perspective and deal with many-to-many and one-to-many relationships. Besides, classical multiple-instance problems are circumvented by correlating events using the data model. The declarative nature of the behavioral perspective makes it possible to model behavioral constraints over activities like cardinality constraints in data models. The object-centric behavioral constraint model created by this novel language is able to describe processes involving interacting instances and complex data dependencies.

This paper mainly focuses on the implementation for modeling business processes using this novel language. We have developed a tool to edit object-centric behavioral constraint models. Experiments show that we can now describe business processes with one-to-many and many-to-many relations that would have remained unsolved using conventional process-model notations. In particular, we illustrate the usefulness of this new approach by extracting a concrete example from the ERP/CRM system Dolibarr.

The remainder is organized as follows. Section 2 presents a real-life process from an ERP system to introduce OCBC models. Section 3 illustrates the ingredients of OCBC models. Section 4 a tool which support editing OCBC models. Section 5 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 involves receiving and fulfilling customer requests for goods or services. It has many variants and our example is based on the scenario in Dolibarr.

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.

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.
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 Behaviorai Constraint 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.
3.1 Semantics of Data Cardinality Constraints

In this paper, objects are data elements generated and used by information systems. These are grouped in classes. 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 ♦) and “always” cardinalities (indicated by □).

![Fig. 2. Example of a class model and corresponding object model.](image)

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).

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 \(r_1\). The annotation “□1..∗” on the target side of \(r_1\) indicates that for each object in \(a\), there is always at least one corresponding object in \(b\). “◇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 \(a\).

---

2 □ indicates the constraint should hold at any point in time and ◇ indicates the constraint should hold from some point onwards.

3 For the sake of brevity, we omit redundant cardinalities in the graph. For instance, “□1” implies “◇1” and therefore “◇1” can be removed in this case.
Modeling Business Processes Using Object-Centric Behavioral Constraints

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

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

<table>
<thead>
<tr>
<th>Constraint</th>
<th>Formalization</th>
</tr>
</thead>
<tbody>
<tr>
<td>response</td>
<td>(((\text{before, after}) \in \mathbb{N} \times \mathbb{N} \mid \text{after} \geq 1))</td>
</tr>
<tr>
<td>unary-response</td>
<td>(((\text{before, after}) \in \mathbb{N} \times \mathbb{N} \mid \text{after} = 1))</td>
</tr>
<tr>
<td>non-response</td>
<td>(((\text{before, after}) \in \mathbb{N} \times \mathbb{N} \mid \text{after} = 0))</td>
</tr>
<tr>
<td>precedence</td>
<td>(((\text{before, after}) \in \mathbb{N} \times \mathbb{N} \mid \text{before} \geq 1))</td>
</tr>
<tr>
<td>unary-precedence</td>
<td>(((\text{before, after}) \in \mathbb{N} \times \mathbb{N} \mid \text{before} = 0))</td>
</tr>
<tr>
<td>non-precedence</td>
<td>(((\text{before, after}) \in \mathbb{N} \times \mathbb{N} \mid \text{before} = 0))</td>
</tr>
<tr>
<td>co-existence</td>
<td>(((\text{before, after}) \in \mathbb{N} \times \mathbb{N} \mid \text{before} + \text{after} \geq 1))</td>
</tr>
<tr>
<td>non-co-existence</td>
<td>(((\text{before, after}) \in \mathbb{N} \times \mathbb{N} \mid \text{before} + \text{after} = 0))</td>
</tr>
</tbody>
</table>

Figure 2(b) shows an object model. The objects are depicted as grey dots: \(\text{Obj} = \{a_1, a_2, b_1, b_2, b_3, c_1, c_2\}\). 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 Semantics of 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 [7].

Figure 3 shows two example behavioral constraints: \(\text{con}_1\) and \(\text{con}_2\). Each constraint corresponds to one constraint type. Table 1 shows eight examples of constraint types. Constraint \(\text{con}_1\) is a response constraint and constraint \(\text{con}_2\) 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 \(\text{con}_{56}\) 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 \(\geq 0\) and after \(\geq 1\) for response). For instance, constraint \(\text{con}_1\) specifies that each \(A\) event should be succeeded by at least one corresponding \(B\) event and constraint \(\text{con}_2\) specifies that each \(B\) event should be succeeded by precisely one \(C\) event.
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.

Fig. 5. An arrow with two reference events (●) can be used as a shorthand. Constraint con\textsubscript{34} (con\textsubscript{56}) corresponds to the conjunction of constraints con\textsubscript{3} and con\textsubscript{4} (resp. con\textsubscript{5} and con\textsubscript{6}).

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\textsubscript{1} and con\textsubscript{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\textsubscript{2} is B (see dot) and the target activity of con\textsubscript{2} is C. The shape (e.g., a double-headed arrow) of each constraint indicates the constraint type. For instance, con\textsubscript{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 Semantics of Object-Centric Behavioral Constraints

Section ?? focused on structuring objects and formalizing cardinality constraints on object models (i.e., classical data modeling) while Section ?? 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.

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 con-
Figure 6. An example model illustrating the main ingredients of OCBC models.

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).
line objects) which are related to c object (i.e., delivery objects) through r2 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 B1 for cr2. Note that, indicated by the example model, B1 refers to precisely one b object that is related to one c object, which means B1 has precisely one target event.

4 OCBC Model Editor

In this part, we introduce how to draw OCBC models (i.e., nodes and edges) using OCBC Model Editor as shown in Fig.7 as well as other functionality provided by the editor.

4.1 Editing Models

This part introduces how to add and delete nodes and edges as well as modifying the names and cardinalities.

**Drawing nodes:** Panel 1 contains two basic elements (i.e., ActivityCell and ClassCell symbols) for drawing activities and classes of OCBC models. One can drag an element to the canvas (i.e., panel 2) and release it, which automatically creates a corresponding node (activity or class) at the point where the element is released. If the element is dragged and release in other panels, nothing is created. As shown in Fig. 8, two activity nodes (“create order” and “create shipment”) and three class nodes (“order”, “order line” and “shipment line”) are created.

**Drawing edges:** Panel 2 supports drawing edges (activity, class and AOC relations) between nodes in an implicit way, i.e., there are no visible symbols for drawing edges. More precisely, when the cursor of the mouse is located at the center of one node (meanwhile a green square appears around the node), an edge can be dragged out from the node (called the source node) for connecting other nodes. If the cursor is released on another node (called the target node), one new edge connecting the source node and target node is generated. If the cursor is released in the blank, a new edge is generated as well as a target node which is the copy of the source node. Note that, there are implicit nodes (denoted by green rectangles) in the middle of activity and class edges, which are used to draw crel relations (i.e., an edge connect an activity edge and a class edge or node) in the similar way. The implicit nodes can be hided to simply the model by checking out the “Connector” in the pop up menu of clicking right button of the mouse. The types (i.e., activity, class, AOC or crel edges) of edges are automatically determined by the source node and target node. For instance, if the source node is an activity and the target node is an class, the edge is an AOC relation. As shown in Fig. 8, the example model has two class edges (denoted by directional lines from source nodes to target nodes), three AOC edges (denoted by dotted lines with small space), one activity edge and one crel edge (denoted by dotted lines with large space).

**Editing nodes and edges:** By double clicking nodes and edges, one can edit their names. The cardinalities of AOC and class edges can be edited by the four pull-down boxes in 3 in Fig. 9. More precisely, when one AOC or class edge is selected, its corresponding boxes are enabled. The cardinalities can be modified by choosing a value,
Fig. 7. The interface of OCBC Model Editor.
or inputting a value and pressing the “Enter” button in the corresponding boxes. The first and second pull-down boxes correspond to the “always” (denoted by “□”) and “eventually” (denoted by “♦”) cardinalities, respectively, on the source (or activity) side of a class (or AOC) edge. The third box corresponds to the “always” cardinality on the target side of a class edge and the cardinality on the class side of an AOC edge, while the fourth box is only used to edit the “eventually” cardinality on the target side of a class edge. For instance, as shown in Fig. 8, the AOC edge (highlighted in red) between “create_order” and “order” is selected, and its cardinalities are the value we set in the first three boxes. Note that it is also possible to directly click the cardinality symbols (e.g., “□1♦1”) and edit cardinalities in the corresponding boxes. Moreover, the constraint type of activity edges can be edited in the pull-down box in Fig. 9 in a similar way. Currently by default, the box provides some single constraint types (such as “unary_response” and “unary_precedence”) and combined constraint types (such as “unary_response-unary_precedence”). It is possible to swap the source node and target node of one constraint by clicking “Swap S/T” in the pop up menu after clicking right button of the mouse.

**Deleting elements:** One can delete a node or an edge by the following three ways after selecting it in the graph:

- press the “Delete” button;
- click the icon similar to “X” in panel (5);
- click the “Delete” item in the pop up menu after clicking right button of the mouse.
Note that, with deleting a node, all edges connected to this node are deleted at the same time. Besides, when one activity edge or class edge is deleted, the related crel edge (if any) is also deleted. We strongly recommend not to delete cardinalities and implicit nodes on edges, which are violating the completeness of models.

4.2 Adjusting the Layout of Models

After editing an OCBC model, it is possible to adjust the layout of the model. One can relocate a class or activity node through dragging its margin to the target location. Note that, if dragging the center of nodes, new edges are created rather than relocating nodes. One edge can be moved through moving its two nodes. Besides, the middle part between its two ends can be adapted to make the model more easy to read, e.g., avoiding overlapping between nodes and edges. To do this, we need to identify the style of an edge we want to adapt. The style of an edge can be modified through the pull-down box in in Fig. 9. There are three common styles, “line”, “elbowEdgeStyle” and “segmentEdgeStyle”. By default, the style is “elbowEdgeStyle”. When one edge is clicked, the box is editable; otherwise, it is disabled. We can see the current style or modify the style of one edge by the box. An edge of the “line” style is a straight line while an edge of the “elbowEdgeStyle” style or “segmentEdgeStyle” style can has one or multiple points which can be dragged to adjust the layout of the edge.

Besides adjusting the layout in a manual way, the edit provide two algorithms to automatically create a proper layout of models. More precisely, in the “Layout” menu, we can find to items: “Vertical Hierarchical” and “Horizontal Hierarchical”. The former one ranks the nodes edges in the vertical direction while the latter one does it in the horizontal direction.

Moreover, the editor supports to zoom in and out models. More precisely, there are three ways to realize this:

- press the “Ctrl” button and roll the wheel of the mouse in panel 2;
- roll the wheel of the mouse in panel 4;
- set a specific value in 3 in panel 5 or use the functionality provided in the “zoom” menu;

Note that the contents shown in panel 2 is the enlarged view of the contents enclosed by the blue rectangle in 4. In other words, we can use the blue rectangle to select a part of the model in panel 4 and view the details in panel 2. The center of the enlarged view is always exactly the center of the blue rectangle by default. We can change the “fixed point” by checking/canceling the “Center Zoom” and “Zoom to Selection” items in the “Zoom” menu. For instance, if the “Zoom to Selection” item is checked and one element of the model is selected, the selected element always appears in the enlarged view no matter we zoom in or out. If the “Center Zoom” is canceled and no element is selected, the “fixed point” is the top left point of the blue rectangle.

4.3 Simplifying Models

When a model is very complex, i.e., containing a lot edges, one can use the “View” menu to hide less important types of edges to simplify the view (by default, all types
of edges are shown). For instance, we can cancel the “AOC relations” to disappear all AOC relations in the model. Note that, we split activity relations into a further level based on constraint types. i.e., we can filter activity relations of a specific constraint type. In the same way, we can also hide the cardinalities and edge names by canceling corresponding items if necessary.

Besides filtering functionality, one can set the size of canvas in panel (2). The smallest unit to make up the canvas is one page which is of the A4 size. The default size of the canvas is two combined page, i.e., the vertical length is twice the width of the A4 page while the horizontal length is the length of the A4 page. One can click the “X/Y page count” item to modify the canvas size by first filling in the “horizontal page count” field and then filling in the “vertical page count” field. If you input 2 in the first field and 1 in the second field, it is exactly the same as the default size.

Additionally, one can configure if the graph shows the grid or page line by checking the “Grid” or “Page breaks” items, respectively.

4.4 File Operation
At any moment of editing a model, it is possible to save the model (with a assigned name) into a XML file (with a suffix “.ocbcemml”) on the local disk. Besides exporting models, importing models is also supported. All these functionality can be realized by “File” menu or corresponding icons in panel (5). We do not give details about how to import and save models because the usage is total the same as common software.

When the editor is launched, it provide a default canvas. It is possible to create multiple canvases using the “New” functionality. Models in different canvases can be saved in separate files. Besides, the interaction between multiple canvases is supported, such as copying nodes from one canvas to another canvas. This makes it easier to create a large model by reusing contents from several small models.

Note that, a strong and useful functionality is provided by this editor, which is to export vector diagrams (i.e., svg and pdf files) of models through selecting corresponding file types in the “Files of Type” pull-down box in the “Save” dialog box.

4.5 Adding attributes
The nodes and edges of OCBC models can have attributes. Accordingly, panel (3) supports to add and remove attributes for a selected element in panel (2).

4.6 Other Functionality
Besides the functionality introduce above, the editor support “Undo” and “Redo” operations. These can be realized through the corresponding icons in panel (5) or buttons such as the combination of “Ctrl” and “Z”. Note that, no all operations can be reverted through the ‘Undo” functionality.

The “Format” menu is used to customize the features of nodes and edges. For instance, through specifying a value for “Opacity”, one can control the transparency of selected nodes or edges. The “Window” menu can modify the window style of the editor and the “Help” menu provide some information about the editor and the link to the manual of the editor.
5 Conclusion

This paper introduces a novel modeling language, named object-centric behavioral constraint language and a tool to support designing models using this language. We illustrate the semantics of OCBC models through a real-life example and explain the functionality of the editor part by part.

In summary, the novel language can describe one-to-many and many-to-many relations in object-centric business processes and reveal data and control-flow perspectives in a single diagram. Besides, the editor has abundant functionalities (in the top menu and the pop up menu) to support editing, zooming in or out, importing and exporting OCBC models. The functionalities are easy to find since they are located in a common way used by other popular software.