Modeling and Reasoning over Declarative Data-Aware Processes with Object-Centric Behavioral Constraints

Alessandro Artale[1], Alisa Kovtunova[1], Marco Montali[1], and Wil M.P. van der Aalst[2]

[1] Free University of Bozen-Bolzano, Italy
surname@inf.unibz.it
[2] Process and Data Science, RWTH Aachen University, Germany
wvdaalst@pads.rwth-aachen.de

Abstract. Existing process modeling notations ranging from Petri nets to BPMN have difficulties capturing the data manipulated by processes. Process models often focus on the control flow, lacking an explicit, conceptually well-founded integration with real data models, such as ER diagrams or UML class diagrams. To overcome this limitation, *Object-Centric Behavioral Constraints* (OCBC) models were recently proposed as a new notation that combines full-fledged data models with control-flow constraints inspired by declarative process modeling notations such as DECLARE and DCR Graphs. We propose a formalization of the OCBC model using temporal description logics. The obtained formalization allows us to lift all reasoning services defined for constraint-based process modeling notations without data, to the much more sophisticated scenario of OCBC. Furthermore, we show how reasoning over OCBC models can be reformulated into decidable, standard reasoning tasks over the corresponding temporal description logic knowledge base.

1 Introduction

Despite the plethora of notations available to model business processes, process modelers struggle to capture real-life processes using mainstream notations such as Business Process Model and Notation (BPMN), Event-driven Process Chains (EPC), and UML activity diagrams. All such notations require the simplifying assumption that each process model focuses on a single, explicitly defined *case* notion (also referred to as *process instance*). The discrepancy between the single case view and reality becomes evident when using process mining techniques to reconstruct processes based on the available data [2]. Process mining starts from the available data and, unless one is using a Business Process Management (BPM) or Workflow Management (WFM) system for process execution, explicit case information is typically missing. Process-centric diagrams using BPMN, EPCs, or UML describe the life-cycle of individual cases. When formal languages like Petri nets, automata, and process algebras are used to describe business processes, they tend to model cases in isolation, and the data perspective is secondary or missing completely. Languages like BPMN allow modelers to attach data to processes, but without the possibility to express complex constraints over such data (e.g., cardinality constraints, is-a links, disjointness, covering, etc. as in ER/UML/ORM data models). Mainstream business process modeling notations describe the *lifecycle of one*

type of process instance at a time missing the opportunity to capture the co-evolution of multiple, interacting instances. In particular, complex constraints over data attached to processes *must* influence the behavior of the process itself—e.g., consider the management of different orders, where the evolution of one order impacts on the possible evolutions of the related orders.

Object-Centric Behavioral Constraint (OCBC) [3, 21, 22] models have been proposed as a modeling language that combines ideas from declarative, constraint-based languages like *DECLARE* [1], and from data modeling languages. OCBC allows to: *(i)* describe the temporal interaction between activities in a given process and to attach (structured) data to processes in a *unified framework*; *(ii)* model the *interactions between multiple process instances*, specifically when there is a *one-to-many* or *many-to-many* relationship between them. Fig. 1 illustrates the way in which OCBC models tackle the above two issues. `Register Email` and `Send Invite` are two activities related to object classes `Person` and `Meeting`, respectively. A meeting is organized by many persons, each of which can in turn organize many meetings. The double-headed arrow connecting `Register Email` and `Send Invite` expresses the constraint that an invitation for a meeting can be sent only if at least one organizer of *that* meeting has previously registered her e-mail. Assuming that the object targeted by each activity is indeed a case for that activity, this simple example already contains two distinct case notions (`Person` and `Meeting`) that are intertwined. In conventional notations, this can only be modeled from the viewpoint of one of the two instances: the registration process of a person or the invitation process for a meeting. Taking the latter viewpoint using conventional notations such as BPMN would require to explicitly introduce a loop to handle the registration of one or more persons organizing a meeting. However, this is incorrect because one registration may be followed by many meetings. One-to-many and many-to-many relationships lead to convergence and divergence problems that cannot be handled in notations describing isolated cases.

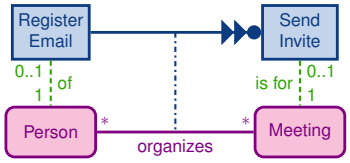


Fig. 1: An OCBC constraint

OCBC models are related to artifact- and data-centric approaches [12, 16, 19] aiming to integrate data and processes. However, this is not done in a single diagram representing different types of process instances and their interactions. In addition, these approaches usually assume complete knowledge over the data, and require to fully spell out data updates when specifying the activities [26, 14]. The few proposals dealing with artifact-centric models with incomplete knowledge [10] do not come with a fully integrated, declarative semantics as done here, but follow instead the Levesque functional approach [20] to separate the evolution of the system from the inspection of (incomplete) knowledge in each state.

This paper provides a complete characterization of the formal semantics of the OCBC approach, unambiguously defining the logical *meaning* of OCBC constraints. We provide a visual and textual syntax for OCBC, then defining the semantics of the different modeling constructs in terms of *temporal description logics*, i.e., a temporal extension of (fragments of) the well-known OWL language. The obtained formalization, in turn, allows us to lift all reasoning services defined for constraint-based process

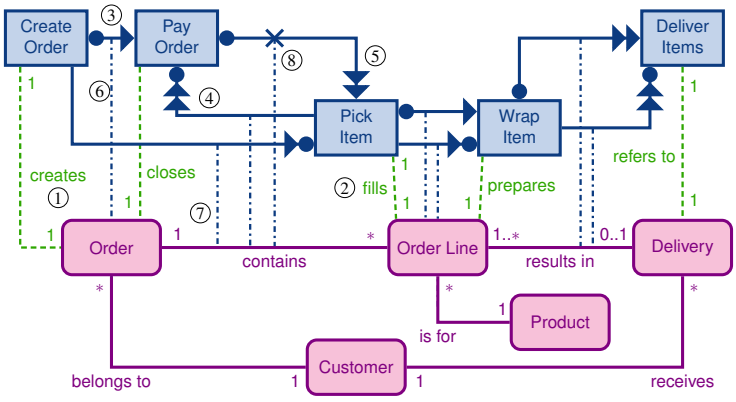


Fig. 2: Example of an OCBC model

modeling notations without data, to the much more sophisticated setting of OCBC. In particular, we show how reasoning over OCBC models can be reformulated into decidable, standard reasoning tasks over the corresponding temporal description logic knowledge base, giving solid foundations to the boundaries of decidability and complexity of reasoning over processes and their manipulated data.

The paper is organized as follows. We present a running example in Sect. 2. Sect. 3 briefly illustrates the temporal DL that will be used to encode and reason over OCBC models. Sect. 4 shows the syntax for OCBC models and their semantics via the temporal DL encoding. Reasoning and verification tasks for OCBC models are tackled in Sect. 5. We present our remarks and future work in Sect. 6.

2 Running Example

The driving assumption underlying our proposal is that processes are modeled as a mirror of their manipulated data. Such data is structured according to complex data modeling constraints (see the lower part of Fig. 2). Data can be attached to activities (see the dotted lines of Fig. 2) and ad-hoc *co-reference* constraints can be expressed on those manipulated data (see the dash-dotted lines of Fig. 2) describing how activities can share/reuse the same data objects.

Example 1. Fig. 2 shows an OCBC model for a process composed by five activities (`CreateOrder`, `PickItem`, `WrapItem`, `PayOrder` and `DeliverItems`) and five object classes in the data model (`Order`, `OrderLine`, `Delivery`, `Product` and `Customer`). The top part describes the temporal ordering of activities and the bottom part how objects relevant for the process execution are structured (read the lower part as a standard UML class diagram). The middle layer (dotted lines) relates activities and data. We now informally describe the constructs highlighted in Fig. 2. ① There is a one-to-one correspondence between a `CreateOrder` activity and an `Order`, i.e., the execution of a `CreateOrder` activity creates a unique `Order` and, vice-versa, due to the 1

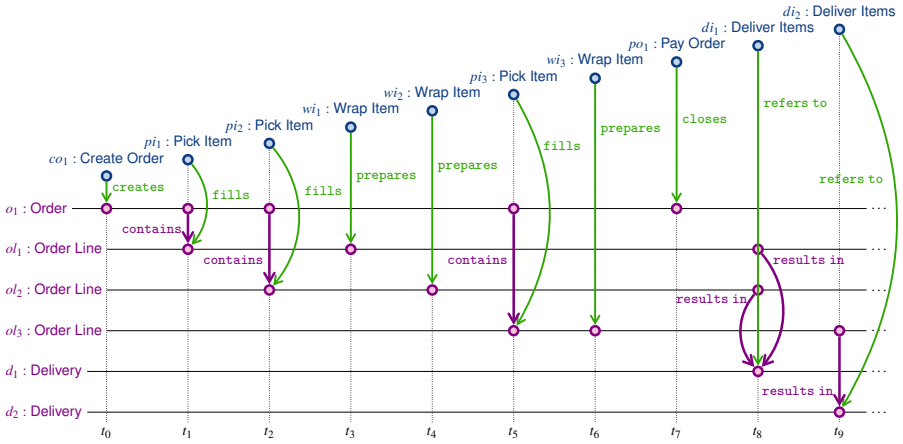


Fig. 3: Trace fragment for the OCBC model in Fig. 2

on the `CreateOrder` side, each `Order` has been *generated* by a single execution of a `CreateOrder` activity. ② Every execution of the `PickItem` activity refers to a unique `OrderLine` and each `OrderLine` has been *generated by* an execution of a `PickItem` activity (and not by a `WrapItem` activity). ③ Each `CreateOrder` activity is followed by exactly one (single arrow) `PayOrder` activity related to the same order. ④ Each `PayOrder` activity is preceded by possibly many (double arrow) `PickItem` activities. ⑤ Whenever we execute `PayOrder` we will never execute `PickItem` on the same paid order. ⑥ The dash-dotted line denotes a *co-reference constraint* over an object class, imposes that when the `CreateOrder` creates an order instance, *that* order instance will eventually be paid by executing a `PayOrder` activity. ⑦ The dash-dotted line is, in this case, a *co-reference constraint* now over a relationship which imposes that when we fill an order line it must have been contained in exactly one order created by executing a `CreateOrder` activity. Since an order line instance could not exist at the same time we create an order instance and relationships are instantiated by co-existing objects, the UML model correctly specifies that, at each point in time, each order participates zero or more times in the `contains` relation. On the other hand, the co-reference constraint together with the mandatory cardinalities constraints and the temporal constraints between `CreateOrder`, `PayOrder` and `PickItem` imply the *eventual* existence of *at least one* order line contained in any given order. ⑧ The dash-dotted line starting with a $\times$ denotes a *negative co-reference constraint* that forbids filling with further order lines an order that has been closed by a `PayOrder` activity.

A possible execution of an OCBC process, called in the following *trace fragment*, records at once events, with their execution time, and the objects they operate on. In addition, it also captures facts that are known to hold over such objects in a given timestamp, in particular, the classes to which objects belong to at that time, as well as how objects are related to each other. In addition, the trace fragment captures, as customary in a standard first-order logic setting, *incomplete knowledge* about a process execution, and OCBC constraints are hence interpreted under the *open-world semantics*. This means that a trace fragment conforms to an OCBC model if it can be extended

towards a full trace that satisfies all the constraints contained therein. A trace fragment conforming to the OCBC model of Fig. 2 is depicted in Fig. 3 and shown in the following first-order logic notation (but also as a DL ABox after a small transformation). We abbreviate activity names with their initials. Instances of activities, classes and relationships are timestamped denoting the execution time of the activity, and the time point when the described fact holds (timestamps respect the time ordering starting from t_0).

$$\texttt{CO}(co_1,t_0),\texttt{PI}(pi_1,t_1),\texttt{PI}(pi_2,t_2),\texttt{WI}(wi_1,t_3),\texttt{WI}(wi_2,t_4),\texttt{PI}(pi_3,t_5),\texttt{WI}(wi_3,t_6),\texttt{PO}(po_1,t_7),$$
$$\texttt{DI}(di_1,t_8),\texttt{DI}(di_2,t_9),\texttt{creates}(co_1,o_1,t_0),\texttt{fills}(pi_1,ol_1,t_1),\texttt{contains}(o_1,ol_1,t_1),\texttt{fills}(pi_2,ol_2,t_2),$$
$$\texttt{contains}(o_1,ol_2,t_2),\texttt{prepares}(wi_1,ol_1,t_3),\texttt{prepares}(wi_2,ol_2,t_4),\texttt{fills}(pi_3,ol_3,t_5),$$
$$\texttt{contains}(o_1,ol_3,t_5),\texttt{prepares}(wi_3,ol_3,t_6),\texttt{closes}(po_1,o_1,t_7),\texttt{refers to}(di_1,d_1,t_8),$$
$$\texttt{results in}(ol_1,d_1,t_8),\texttt{results in}(ol_2,d_1,t_8),\texttt{refers to}(di_2,d_2,t_9),\texttt{results in}(ol_3,d_2,t_9),$$

The process described in the example cannot be modeled using conventional process modeling languages, because (a) three different types of instances (of activities, classes and also relationships instances) are intertwined in a uniform framework so that no further coding or annotations are needed, and (b) cardinality and structural constraints in the object class model influence the allowed behavior of activities, and vice-versa. Take, e.g., the fact that in the example we have three different `OrderLine` instances (ol_1, ol_2, ol_3), then, together with the co-reference constraints on `OrderLine`, we implicitly enforce the occurrence of three different `PickItem` and `WrapItem` activities.

3 A Gentle Introduction to Temporal DLs

Since description logics (DLs) are able to capture data models [11, 4, 17] and are the logical formalism underpinning ontologies expressed in the standard Web Ontology Language OWL (`www.w3.org/2007/OWL`), while the linear temporal logic (LTL) is able to formalize the temporal interweaving of the activities in a process [1], we propose here to use temporal description logics based on $T_{\mathcal{US}}\mathcal{ALCQI}$ and its fragments [27, 18, 8] to formally describe the semantics of OCBC models and to capture in a uniform formalism both the processes and their attached data.

$T_{\mathcal{US}}\mathcal{ALCQI}$ is one of the most expressive and still decidable temporal description logics. The language alphabet contains *object names* $a_0, a_1, \ldots$, *concept names* $A_0, A_1, \ldots$ and *role names* $P_0, P_1, \ldots$. Then, *roles* R and *concepts* C are given by the following grammar:

$$R ::= P_i \mid R^- \qquad\qquad C ::= \top \mid A_i \mid (\geq q\, R\, C) \mid \neg C \mid C_1 \sqcap C_2 \mid C_1\, \mathcal{U}\, C_2 \mid C_1\, \mathcal{S}\, C_2$$

where R^- denotes the *inverse* of the role R (obtained by reversing the relation R) and q is a positive integer. We use the standard abbreviations: $C_1 \sqcup C_2 = \neg(\neg C_1 \sqcap \neg C_2)$, $\bot = \neg\top$, $\exists R = (\geq 1\, R\ \top)$, $\exists R.C = (\geq 1\, R\ C)$, $(\leq q\, R\ C) = \neg(\geq (q+1)\, R\ C)$.Furthermore, all the temporal operators used in LTL can be expressed via $\mathcal{S}$ 'since' and $\mathcal{U}$ 'until' [18]. Operators $\Diamond_F$ and $\Diamond_P$ ('sometime in the future/past') can be expressed as $\Diamond_F C = \top\, \mathcal{U}\, C$ and $\Diamond_P C = \top\, \mathcal{S}\, C$; operators $\Box_F$ ('always in the future') and $\Box_P$ ('always in the past') are defined as dual to $\Diamond_F$ and $\Diamond_P$, i.e., $\Box_F C = \neg\Diamond_F\neg C$ and $\Box_P C = \neg\Diamond_P\neg C$. The non-strict operators (including the current evaluation time), denoted as $\Diamond_P^+$ and $\Diamond_F^+$, can be captured as $\Diamond_P^+ C = C \sqcap \Diamond_P C$ and $\Diamond_F^+ C = C \sqcap \Diamond_F C$ (similarly, $\Box_P^+$ and $\Box_F^+$ are defined

as the dual operators of $\Diamond_P^+$ and $\Diamond_F^+$, respectively). The 'always' operator $\boxplus$ can be expressed as $\boxplus C = \Box_F \Box_P C$, while the dual 'sometime' is defined as $\diamondplus C = \neg \boxplus \neg C$. Finally, the temporal operators $\bigcirc_F$ ('next time') and $\bigcirc_P$ ('previous time') can be defined as $\bigcirc_F C = \bot \,\mathcal{U}\, C$ and $\bigcirc_P C = \bot \,\mathcal{S}\, C$.

A $T_{\mathcal{US}}\mathcal{ALCQI}$ *TBox* $\mathcal{T}$ is a finite set of *concept* and *role inclusion* axioms of the form $C_1 \sqsubseteq C_2$ and $R_1 \sqsubseteq R_2$, respectively. An *ABox*, $\mathcal{A}$, consists of assertions of the form $\bigcirc^n A_k(a_i)$, $\bigcirc^n P_k(a_i, a_j)$, where A_k is a concept name, P_k a role name, a_i, a_j object names and, for $n \in \mathbb{Z}$,

$$\bigcirc^n = \underbrace{\bigcirc_F \cdots \bigcirc_F}_{n \text{ times}}, \text{ if } n \geq 0, \text{ and } \bigcirc^n = \underbrace{\bigcirc_P \cdots \bigcirc_P}_{-n \text{ times}}, \text{ if } n < 0.$$

Taken together, the TBox $\mathcal{T}$ and ABox $\mathcal{A}$ form the *knowledge base* (KB) $\mathcal{K} = (\mathcal{T}, \mathcal{A})$. In this paper, OCBC models will be encoded using TBoxes (see Sect. 4.4), while single process executions (i.e., trace fragments as shown in Example 1) are encoded as ABoxes (e.g., $\mathtt{CO}(co_1, t_0)$ is encoded as $\bigcirc^{t_0}\mathtt{CO}(co_1)$).

A *temporal interpretation* is a structure of the form $\mathcal{I} = ((\mathbb{Z}, <), \Delta^{\mathcal{I}}, \{\cdot^{\mathcal{I}} \mid n \in \mathbb{Z}\})$, where $(\mathbb{Z}, <)$ is the linear model of time, $\Delta^{\mathcal{I}}$ is a non-empty interpretation domain and $\mathcal{I}(n)$ gives a standard DL interpretation for each time instant $n \in \mathbb{Z}$: $\mathcal{I}(n) = \big(\Delta^{\mathcal{I}}, a_0^{\mathcal{I}(n)}, A_0^{\mathcal{I}(n)}, \ldots, P_0^{\mathcal{I}(n)}, \ldots\big)$, assigning to each concept name A_i a unary predicate $A_i^{\mathcal{I}(n)} \subseteq \Delta^{\mathcal{I}}$ and to each role name P_i a binary relation $P_i^{\mathcal{I}(n)} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$. We assume that the domain $\Delta^{\mathcal{I}}$ and the interpretations $a_i^{\mathcal{I}} \in \Delta^{\mathcal{I}}$ of object names are the same for all $n \in \mathbb{Z}$, i.e., we adopt the *constant domain assumption* and *rigid designators* (consult [18] for more details on these assumptions). At each time instant $n \in \mathbb{Z}$, role and concept constructs are interpreted as follows

$$\begin{aligned}
(R^-)^{\mathcal{I}(n)} &= \{(y,x) \in \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \mid (x,y) \in R^{\mathcal{I}(n)}\},\\
(\geq q\, R\, C)^{\mathcal{I}(n)} &= \big\{x \in \Delta^{\mathcal{I}} \mid \sharp\{y \in C^{\mathcal{I}(n)} \mid (x,y) \in R^{\mathcal{I}(n)}\} \geq q\big\},\\
(\neg C)^{\mathcal{I}(n)} &= \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}(n)}, \quad \top^{\mathcal{I}} = \Delta^{\mathcal{I}}, \quad (C_1 \sqcap C_2)^{\mathcal{I}(n)} = C_1^{\mathcal{I}(n)} \cap C_2^{\mathcal{I}(n)},\\
(C_1 \,\mathcal{U}\, C_2)^{\mathcal{I}(n)} &= \bigcup_{k>n} \big(C_2^{\mathcal{I}(k)} \cap \bigcap_{n<m<k} C_1^{\mathcal{I}(m)}\big),\\
(C_1 \,\mathcal{S}\, C_2)^{\mathcal{I}(n)} &= \bigcup_{k<n} \big(C_2^{\mathcal{I}(k)} \cap \bigcap_{n>m>k} C_1^{\mathcal{I}(m)}\big),
\end{aligned}$$

where $\sharp X$ denotes the cardinality of X. Thus, for example, $x \in (C_1 \,\mathcal{U}\, C_2)^{\mathcal{I}(n)}$ iff there is a moment $k > n$ such that $x \in C_2^{\mathcal{I}(k)}$ and $x \in C_1^{\mathcal{I}(m)}$, for all moments m between n and k. Note that the operators $\mathcal{S}$ and $\mathcal{U}$ are 'strict' in the sense that their semantics does not include the current moment of time.

Concept and role inclusion axioms (TBox) are interpreted in $\mathcal{I}$ *globally*:

$$\begin{aligned}
\mathcal{I} \models C_1 \sqsubseteq C_2 \quad &\text{iff} \quad C_1^{\mathcal{I}(n)} \subseteq C_2^{\mathcal{I}(n)} \quad \text{for } \textit{all } n \in \mathbb{Z},\\
\mathcal{I} \models R_1 \sqsubseteq R_2 \quad &\text{iff} \quad R_1^{\mathcal{I}(n)} \subseteq R_2^{\mathcal{I}(n)} \quad \text{for } \textit{all } n \in \mathbb{Z}.
\end{aligned}$$

ABox assertions are interpreted relatively to the *initial moment*, 0:

$$\begin{aligned}
\mathcal{I} \models \bigcirc^n A_k(a_i) \quad &\text{iff} \quad a_i^{\mathcal{I}} \in A_k^{\mathcal{I}(n)},\\
\mathcal{I} \models \bigcirc^n P_k(a_i, a_j) \quad &\text{iff} \quad (a_i^{\mathcal{I}}, a_j^{\mathcal{I}}) \in P_k^{\mathcal{I}(n)}.
\end{aligned}$$

We call $\mathcal{I}$ a *model* of a KB $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ and write $\mathcal{I} \models \mathcal{K}$ if $\mathcal{I}$ satisfies all inclusions in $\mathcal{T}$ and all assertions in $\mathcal{A}$. A KB $\mathcal{K}$ is *satisfiable* if it has a model. A concept C (role R) is *satisfiable* with respect to $\mathcal{K}$ if there are a model $\mathcal{I}$ of $\mathcal{K}$ and $n \in \mathbb{Z}$ such that $C^{\mathcal{I}(n)} \neq \emptyset$ (respectively, $R^{\mathcal{I}(n)} \neq \emptyset$). It is readily seen that the concept and role satisfiability problems are equivalent to KB satisfiability.

Reasoning in $T_{\mathcal{US}}\mathcal{ALCQI}$ w.r.t. to a KB is a problem which has been proven to be ExpTime-complete [27, 18]. To achieve better complexity results fragments of $\mathcal{ALCQI}$ must be considered. Nice results have been gained when temporalizing *DL-Lite* logics [13, 6]—see, e.g., the temporal *DL-Lite* called $T_{\mathcal{US}}$*DL-Lite*$_{bool}^{(\mathcal{HN})}$ where reasoning has the same complexity of LTL reasoning, i.e., PSpace-complete [8].

4 The OCBC Model

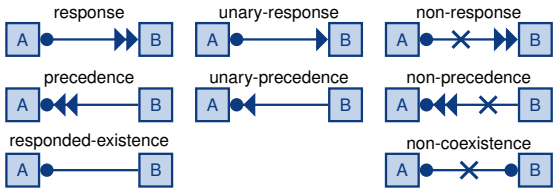
We now present the syntax and graphical appearance of OCBC models, together with their formal semantics. The original proposal of the OCBC model is the way activities and data are related. In particular, an OCBC model captures, at once: *(i) Data dependencies*, represented using standard data modeling constructs, i.e., *classes*, *relationships* and *constraints* between them; *(ii) Activities*, accounting for units of work within a process; *(iii) Mutual relationships between activities and classes*, linking the execution of activities in a given process with the data objects they manipulate; *(iv) Temporal constraints* between activities; *(v) Co-reference constraints* that enforce the application of temporal constraints, and in particular limit their application to those activities that indirectly co-refer thanks to the objects and relationships they point to.

4.1 The Data Model – ClaM

Data used by the activities of an OCBC model is structured according to a standard modeling language, i.e., ER/UML/ORM. While $\mathcal{ALCQI}$ is able to fully capture the semantics of such data models (see [11, 4, 17] and references therein) in the following, just for the sake of simplicity and lack of space, we present only a subset of the complete set of modeling constructs allowed in those standard data modeling languages and denote such set of modeling constructs as the ClaM data model (which stands for *CLAss data Model*). In particular, the following syntax limits ClaM to capture object classes that can be organized along ISA hierarchies (with possibly disjoint sub-classes and covering constraints), binary relationships between object classes and cardinalities expressing participation constraints of object classes in relationships.

Definition 1 (ClaM Syntax). *A conceptual schema Σ in the Class Model, ClaM, is a tuple $\Sigma = (\mathcal{U}_C, \mathcal{U}_R, \tau, \#_{dom}, \#_{ran}, \mathsf{ISA}, \mathrm{DISJ}, \mathrm{COV})$, where:*

- *$\mathcal{U}_C$ is the universe of object classes. We denote object classes as $O_1, O_2, \ldots$;*
- *$\mathcal{U}_R$ is the universe of binary relationships among object classes. We denote relationships as $R_1, R_2, \ldots$;*
- *$\tau : \mathcal{U}_R \to \mathcal{U}_C \times \mathcal{U}_C$ is a* total *function associating a signature to each binary relationship. If $\tau(R) = (O_1, O_2)$ then O_1 is the* range *and O_2 the* domain *of the relationship;*
- *$\#_{dom} : \mathcal{U}_R \times \mathcal{U}_C \nrightarrow \mathbb{N} \times (\mathbb{N} \cup \{\infty\})$ is a* partial *function defining* cardinality constraints *on the domain of a relationship. $\#_{dom}(R, O)$ is defined only if $\tau(R) = (O, O_1)$;*



response(A,B)	If A is executed, then B must be executed afterwards.
unary-response(A,B)	If A is executed, then B must be executed exactly once afterwards.
precedence(A,B)	If A is executed, then B must have been executed before.
unary-precedence(A,B)	If A is executed, then B must have been executed exactly once before.
responded-existence(A,B)	If A is executed, then B must also be executed (either before or afterwards).
non-response(A,B)	If A is executed, then B will not be executed afterwards.
non-precedence(A,B)	If A is executed, then B was never executed before.
non-coexistence(A,B)	A and B cannot be both executed.

Fig. 4: Types of temporal constraints between activities and their intuitive semantics

- $\#_{ran} : \mathcal{U}_R \times \mathcal{U}_C \nrightarrow \mathbb{N} \times (\mathbb{N} \cup \{\infty\})$ *is a* partial *function defining* cardinality constraints *on the range of a relationship.* $\#_{ran}(R,O)$ *is defined only if* $\tau(R) = (O_1, O)$*;*
- $\mathsf{ISA} \subseteq \mathcal{U}_C \times \mathcal{U}_C$ *is a binary relation defining the super-class and sub-class hierarchy on object classes. If* $\mathsf{ISA}(C_1, C_2)$ *then* C_1 *is said to be a* sub-class *of* C_2 *while* C_2 *is said to be a* super-class *of* C_1*;*
- DISJ $\subseteq 2^{\mathcal{U}_C} \times \mathcal{U}_C$ *is a binary relation defining the set of disjoint sub-classes in an* ISA *hierarchy;*
- COV $\subseteq 2^{\mathcal{U}_C} \times \mathcal{U}_C$ *is a binary relation defining the set of sub-classes covering the super-class in an* ISA *hierarchy.*

As for the full-fledged syntax of ER/UML/ORM, their formal set-theoretic semantics, and their translation as $\mathcal{ALCQI}$ KBs we refer to [11, 4, 17]. Concerning the semantics of the ClaM constructs, cardinality constraints are interpreted as the number of times each instance of the involved class participates in the given relationship, ISA is interpreted as sub-setting, DISJ and COV are interpreted in the obvious way using disjointness/union between classes, relationships are interpreted as binary predicates, while the relationship signature acts as a typing for its arguments.

Example 2. The lower part of the OCBC model shown in Fig. 2 captures the data model as a ClaM diagram with:

$$\mathcal{U}_C = \{\texttt{Order}, \texttt{OrderLine}, \texttt{Product}, \texttt{Customer}, \texttt{Delivery}\};$$
$$\mathcal{U}_R = \{\texttt{contains}, \texttt{belongs to}, \texttt{is for}, \texttt{results in}, \texttt{receives}\};$$
$$\tau(\texttt{contains}) = (\texttt{Order}, \texttt{OrderLine}), \ldots$$
$$\#_{dom}(\texttt{contains}, \texttt{Order}) = (0, \infty);\ \#_{ran}(\texttt{contains}, \texttt{OrderLine}) = (1, 1); \ldots$$

Cardinalities are shown in the diagram following the UML reading.

4.2 Temporal Constraints over Activities

Taking inspiration from the DECLARE patterns [1], we present here the temporal constraints between (pairs of) activities that can be expressed in OCBC. Fig. 4 graphi-

cally renders such constraints together with their intuitive meaning. In the following we present their syntax.

Definition 2 (Temporal constraints). *Let*

- $\mathcal{U}_A$ *be the universe of activities, denoted with capital letters* $A_1, A_2, \ldots$;
- $\mathcal{U}_{TC}$ *be the universe of* temporal constraints*, i.e.,* $\mathcal{U}_{TC} = \{$`response`, `unary-response`, `precedence`, `unary-precedence`, `responded-existence`, `non- response`, `non-precedence`, `non-coexistence`$\}$*, where each* $tc \in \mathcal{U}_{TC}$ *is a binary relation over activities, i.e.,* $tc \subseteq \mathcal{U}_A \times \mathcal{U}_A$.

The set of temporal constraints *in a given OCBC model is denoted as* Σ_{TC} *and is conceived as a set of elements of the form* $tc(A_1, A_2)$*, where* $tc \in \mathcal{U}_{TC}$ *and* $A_1, A_2 \in \mathcal{U}_A$.

Remark 1. We observe that the *non-precedence* constraint is syntactic sugar, as it can be emulated using *non-response*: `non-precedence`$(A,B) \equiv$ `non-response`(B,A). Thus, in the following we will not consider it anymore. When defining later on the OCBC model we will consider the set Σ^+_{TC} of *positive* constraints containing `response`, `unary-response`, `precedence`, `unary-precedence`, and `responded-existence`, and the set Σ^-_{TC} of *negative* constraints containing `non-response` and `non-coexistence`.

4.3 Syntax of OCBC Models

We are now ready to define the OCBC model starting from data models and temporal constraints as respectively defined in Sections 4.1 and 4.2.

Definition 3 (OCBC syntax). *An OCBC model,* $\mathcal{M}$*, is a tuple:*
$(ClaM, \Sigma_{TC}, \mathcal{U}_A, \mathcal{U}_{R_{AC}}, \tau_{R_{AC}}, \#_{act}, \#_{obj}, cref, neg\text{-}cref)$*, where:*

- *ClaM is a data model as in Def. 1, and* Σ_{TC} *a set of temporal constraints as in Def. 2;*
- $\mathcal{U}_A$ *is the universe of activities;*
- $\mathcal{U}_{R_{AC}}$ *is the universe of* activity-object relationships *being a set of binary relationships;*
- $\tau_{R_{AC}} : \mathcal{U}_{R_{AC}} \to \mathcal{U}_A \times \mathcal{U}_C$ *is a* total *function associating a signature to each activity-object relationship. If* $\tau_{R_{AC}}(R) = (A, O)$ *then* $A \in \mathcal{U}_A$ *and* $O \in \mathcal{U}_C$*;*
- $\#_{act} : \mathcal{U}_{R_{AC}} \times \mathcal{U}_A \nrightarrow \mathbb{N} \times (\mathbb{N} \cup \{\infty\})$ *is a* partial function *defining* cardinality constraints *on the participation of activities in activity-object relationships.* $\#_{act}(R, A)$ *is defined only if* $\tau_{R_{AC}}(R) = (A, O)$*;*
- $\#_{obj} : \mathcal{U}_{R_{AC}} \times \mathcal{U}_C \nrightarrow \{1\}$ *is a* partial function *denoting the activity that* generated *a given object in O.* $\#_{obj}(R, O)$ *is defined only if* $\tau_{R_{AC}}(R) = (A, O)$*;*
- *cref is the partial function of* co-reference constraints *s.t.*
$$cref : \Sigma^+_{TC} \times \mathcal{U}_{R_{AC}} \times \mathcal{U}_{R_{AC}} \nrightarrow \mathcal{U}_C \cup \mathcal{U}_R;$$
- *neg-cref is the partial function of* negative co-reference constraints *s.t.*
$$neg\text{-}cref : \Sigma^-_{TC} \times \mathcal{U}_{R_{AC}} \times \mathcal{U}_{R_{AC}} \nrightarrow \mathcal{U}_C \cup \mathcal{U}_R.$$

Inverses of activity-object relationships are assumed to be functional capturing the intuition that a single occurrence of an activity can manipulate an object at a given point in time. To clarify the syntax of the OCBC modeling language we illustrate the scenario provided in Example 1.

Example 3. We consider the OCBC model in Fig. 2 where the activities are depicted in the upper part of the figure while the lower part shows the ClaM data model for the data manipulated by the activities of the process. The set $\mathcal{U}_{R_{AC}}$ of the *activity-object relationships* is: $\mathcal{U}_{R_{AC}} = \{\texttt{create}, \texttt{closes}, \texttt{fills}, \texttt{prepares}, \texttt{refers to}\}$ connecting an activity with the manipulated objects as an effect of executing the activity itself. For example, the activity `CreateOrder` `creates` an instance of the object class `Order` when it is executed. Cardinality constraints can be added to activity-object relationships to specify participation constraints either on the activity side or on the object class side. For example, each execution of `PickItem` `fills` one and only one `OrderLine`, i.e., $\#_{act}(\texttt{fills}, \texttt{PickItem}) = (1,1)$. On the other hand, any `OrderLine` must be necessarily filled by executing a `PickItem` activity, i.e., $\#_{obj}(\texttt{fills}, \texttt{OrderLine}) = 1$. The *co-reference constraints* involving object classes specify constraints on how objects connected to different activities can be shared. For example, the `OrderLine` instance filled by a `PickItem` is the same as the one prepared by the corresponding `WrapItem`. These co-reference constraints can be expressed using the following OCBC syntax:

$$\textit{cref}\big(\texttt{unary-response}(\texttt{PickItem}, \texttt{WrapItem}), \texttt{fills}, \texttt{prepares}\big) = \texttt{OrderLine},$$
$$\textit{cref}\big(\texttt{unary-precedence}(\texttt{WrapItem}, \texttt{PickItem}), \texttt{prepares}, \texttt{fills}\big) = \texttt{OrderLine}.$$

The co-reference constraint ⑦, and the negative co-reference constraint ⑧ are expressed as, respectively:

$$\textit{cref}(\texttt{unary-precedence}(\texttt{PickItem}, \texttt{CreateOrder}), \texttt{fills}, \texttt{creates}) = \texttt{contains};$$
$$\textit{neg-cref}(\texttt{non-response}(\texttt{PayOrder}, \texttt{PickItem}), \texttt{closes}, \texttt{fills}) = \texttt{contains}.$$

4.4 Semantics of OCBC Models

We now focus on the semantics of OCBC models. As pointed out in Sect. 2, OCBC models are interpreted using traces that capture the occurrence of events, the relationships between events and objects, and the evolution of objects and relationships over time. Here, we base the OCBC semantics on *infinite* traces (cf. Sect. 6 for a remark on finite traces). The information recorded in an actual execution trace is interpreted under incomplete knowledge, i.e., as a *trace fragment* containing explicit factual knowledge that is known to certainly hold but, in general, only partially capturing what actually occurred. Thus, the notion of trace as used in event log formats such as the XES IEEE standard has to be interpreted, in our setting, as a trace fragment.

Our effort is to reconcile the process flow semantics with the data model semantics. We thus resort to a knowledge base expressed in the temporal DL $T_{\mathcal{US}}\mathcal{ALCQI}$. In particular, we map both activities and object classes to $T_{\mathcal{US}}\mathcal{ALCQI}$ concepts, while activity-object relationships and relationships of the data model are mapped to $T_{\mathcal{US}}\mathcal{ALCQI}$ roles. Such an encoding of OCBC models using KBs in the temporal DL $T_{\mathcal{US}}\mathcal{ALCQI}$ interprets constraints of an OCBC model over infinite traces, while the ABox, that encodes the explicit factual knowledge, i.e., the *trace fragment* at hand, is interpreted as a finite portion of such infinite traces. Here we detail the encoding.

Concerning the semantics of the *ClaM data model*, we interpret it via a mapping to $\mathcal{ALCQI}$ as already discussed in Sect. 4.1. Furthermore, we can add to the data model temporal constraints captured in $T_{\mathcal{US}}\mathcal{ALCQI}$ as shown in [5, 7].

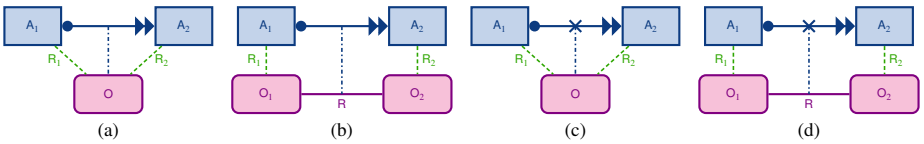


Fig. 5: Co-reference (response) constraints over (a) object classes and (b) relationships, with their negated versions (c-d)

As for *activity-object relationships*, let $R \in \mathcal{U}_{R_{AC}}$ so that $\tau_{R_{AC}}(R) = (A, O)$. The following $T_{\mathcal{US}}\mathcal{ALCQI}$ axioms captures *inverse functionality*, and *domain* and *range* restrictions for R:

$$(\geq\ 2\ R^-\ \top) \sqsubseteq \bot, \qquad \exists R \sqsubseteq A, \quad \exists R^- \sqsubseteq O. \tag{1}$$

A cardinality constraint of the form $\#_{obj}(R, O) = 1$, denoting the activity that generated an object of class O, is captured as:

$$O \sqsubseteq \Diamond_P^+ (O \sqcap \exists R^-).$$

Cardinality constraints for the participation of activities in activity-object relationships ($\#_{act}$) are instead captured as classical cardinalities in data models (see [11, 5, 7]).

Semantics of co-reference constraints. Having fixed the semantics for the ClaM data model and the one for the activity-object relationships we are left with the most tricky aspect of OCBC, namely the semantics of *co-reference constraints*. In the following, we consider the different kinds of co-reference constraints which, according to Definition 3, can be either positive or negative, and can range either over object classes (as illustrated in Fig. 5a and 5c) or over relationships (as illustrated in Fig. 5b and 5d). Let $R_1, R_2 \in \mathcal{U}_{R_{AC}}$, $A_1, A_2 \in \mathcal{U}_A$ and $O \in \mathcal{U}_C$ s.t. $tc(A_1, A_2) \in \Sigma_{TC}^+$, $\tau_{R_{AC}}(R_1) = (A_1, O)$, $\tau_{R_{AC}}(R_2) = (A_2, O)$ and *cref* be a *co-reference constraint over object classes* of the form: $cref(tc(A_1, A_2), R_1, R_2) = O$ (as in Fig. 5a). Then, *co-reference over object classes* when *tc* is the `response` temporal constraint is captured by the axiom:

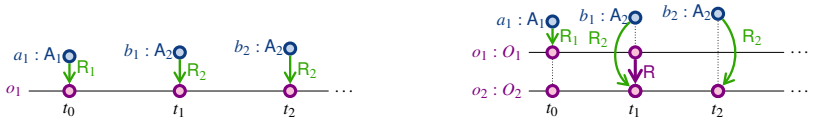
$$\exists R_1^- \sqsubseteq \Diamond_F \exists R_2^- \tag{2}$$

This expresses that "*whenever an object is in the range of* R_1 *then sometime in the future it must be also in the range of* R_2". This semantics enforces a temporal constraint over the activities via the co-referenced object, i.e., when the activity A_1 is linked via R_1 to an object in O then it must be followed by an execution of A_2 referencing the *same* object via R_2. Formally, the following logical implication holds:

$$\{(1), (2), A_1 \sqsubseteq \exists R_1\} \models A_1 \sqsubseteq \exists R_1 . \Diamond_F \exists R_2^- . A_2 \tag{3}$$

When *tc* is the `unary-response` temporal constraint we need to add to formula (2) another formula that guarantees a *unique* occurrence of A_2 over the co-referenced object:

$$\exists R_2^- \sqcap \Diamond_P \exists R_1^- \sqsubseteq \Box_F \neg \exists R_2^- \tag{4}$$



(a) (b)

Fig. 6: (a) Trace fragment for (2) but not (4); (b) Trace fragment for (8) but not (10)

Fig. 6a shows a possible instantiation of the OCBC model in Fig. 5a which, in turn, is not a valid fragment in case the temporal constraint is changed to `unary-response`. Similar formulas hold when *tc* is a temporal constraint over the past, i.e., either `precedence` (formula (5)), `unary-precedence` (formulas (5) and (6)) or `responded-existence` (formula (7)).

$$\exists R_1^- \sqsubseteq \Diamond_P \exists R_2^- \tag{5}$$

$$\exists R_2^- \sqcap \Diamond_F \exists R_1^- \sqsubseteq \Box_P \neg \exists R_2^- \tag{6}$$

$$\exists R_1^- \sqsubseteq \Diamond\!\!\!\!- \exists R_2^- \tag{7}$$

We now consider *co-reference constraints over relationships*. As in Fig. 5b, let $O_1, O_2 \in \mathcal{U}_C$, $R \in \mathcal{U}_R$, with $\tau(R) = (O_1, O_2)$, $\tau_{R_{AC}}(R_1) = (A_1, O_1)$, $\tau_{R_{AC}}(R_2) = (A_2, O_2)$ and *cref* be a co-reference of the form: $cref(tc(A_1, A_2), R_1, R_2) = R$. Then, the semantics of *co-reference over relationships* when *tc* is the `response` constraint is captured by:

$$\exists R_1^- \sqsubseteq \Diamond_F \exists R. \exists R_2^- \tag{8}$$

Expressing that "*every object in the range of R_1 sometime in the future should be connected via R to an object in the range of R_2*." A logical implication similar to (3) holds:

$$\{(1), (8), A_1 \sqsubseteq \exists R_1\} \models A_1 \sqsubseteq \exists R_1 . \Diamond_F \exists R. \exists R_2^- . A_2 \tag{9}$$

When *tc* is `unary-response` we should add to formula (8) another formula that guarantees that activity A_1 is followed by a single occurrence of A_2 via R. The following axiom expresses that "*whenever an object is in the range of R_2 (thus under the occurrence of A_2) and is connected via R^- to an object that before was in the range of R_1 (due to the occurrence of the activity A_1) then, it will never be in the range of R_2.*"

$$\exists R_2^- \sqcap \exists R^- . \Diamond_P \exists R_1^- \sqsubseteq \Box_F \neg \exists R_2^- \tag{10}$$

Fig. 6b shows an instantiation of the OCBC model in Fig. 5b that, in turn, is not anymore a valid fragment in case the temporal constraint is changed to `unary-response` (because o_2 is pointed to by two different instances—b_1, b_2—of the activity A_2). Similar formulas hold when *tc* is `precedence` (axiom (11)), `unary-precedence` (axioms (11) and (12)) and `responded-existence` (axiom (13))

$$\exists R_1^- \sqsubseteq \exists R. \Diamond_P \exists R_2^- \tag{11}$$

$$\exists R_2^- \sqcap \Diamond_F \exists R^- . \exists R_1^- \sqsubseteq \Box_P \neg \exists R_2^- \tag{12}$$

$$\exists R_1^- \sqsubseteq \Diamond\!\!\!\!- \exists R. \Diamond\!\!\!\!- \exists R_2^- \tag{13}$$

Note that axiom (13) allows for `responded-existence` to be symmetric—as for axiom (7)—i.e., $\{(13)\} \models \exists R_2^- \sqsubseteq \Diamond \exists R^- . \Diamond \exists R_1^-$.

We now consider co-references in the presence of *negative behavioral constraints* (see Fig. 5c-5d). We start with co-reference over object classes. In case *tc* is `non-response` (as in Fig. 5c) then the following axiom expresses that "*whenever an object is in the range of R_1 then never in the future it could be in the range of R_2*":

$$\exists R_1^- \sqsubseteq \Box_F \neg \exists R_2^- . \tag{14}$$

As a consequence of this axiom, and of the fact that the domains of R_1 and R_2 are activities A_1 and A_2, while they both range over the same class O, we can also read this negative co-reference as "*every instance of activity A_1 can never be followed by instances of A_2 sharing the same object in O*". The right-hand side of the axiom is the negation of the right-hand side of axiom (2). When *tc* is `non-coexistence`, we have

$$\exists R_1^- \sqsubseteq \circledast \neg \exists R_2^- \tag{15}$$

Again, the right-hand side is the negation of the right-hand side of axiom (7).

When negative co-references involve a relationship and *tc* is `non-response` (as in Fig. 5d) the following axiom expresses that "*whenever an object is in the range of R_1 then never in the future it could be connected via R to an object in the range of R_2 (thus under the occurrence of A_2)*":

$$\exists R_1^- \sqsubseteq \Box_F \neg \exists R . \exists R_2^- \tag{16}$$

implying that "*every instance of activity A_1 can never be followed by instances of A_2 sharing the same pair of objects in R*". Notice again that the right-hand side of the above axiom is the negation of the right-hand side of axiom (8). Finally, by negating the right-hand side of axiom (13) we capture the case when *tc* is `non-coexistence`

$$\exists R_1^- \sqsubseteq \circledast \neg \exists R . \Diamond \exists R_2^- \tag{17}$$

Similar to `responded-existence`, `non-coexistence` over both object classes (15) and relationships (17) is obviously symmetric. Formally, considering the co-reference over a relationship, $\{(17)\} \models \exists R_2^- \sqsubseteq \circledast \neg \exists R^- . \Diamond \exists R_1^-$.

Altogether, an OCBC model can be captured via a TBox in $T_{\mathcal{US}}\mathcal{ALCQI}$, and its trace fragments using corresponding ABoxes. Overall, a $T_{\mathcal{US}}\mathcal{ALCQI}$ KB is thus able to provide a uniform representation for OCBC, on which we can apply ad hoc reasoning services as described in the following section.

5 Verification and Reasoning over OCBC Models

The main motivation to provide a mapping from OCBC models to a DL Knowledge Base is the possibility of carrying out automated reasoning over them. We discuss how the typical services for verifying declarative, constraint-based process models can be lifted to the more sophisticated setting of OCBC. To do so, we build on the services defined for the well-established DECLARE language [25, 24]. In the following, we show

how such services can be reformulated as standard reasoning tasks over $T_{US}\mathcal{ALCQI}$ knowledge bases, in turn inheriting their decidability and worst-case complexity.

Let $\mathcal{M}$ be an OCBC model of interest, and ρ a trace fragment over $\mathcal{M}$. We denote by $\mathcal{T}_\mathcal{M}$ and $\mathcal{A}_\rho$ the TBox and ABox obtained by encoding $\mathcal{M}$ and ρ in $T_{US}\mathcal{ALCQI}$, and by $\mathcal{K}_{\mathcal{M},\rho}$ the resulting $T_{US}\mathcal{ALCQI}$ KB, i.e., $\mathcal{K}_{\mathcal{M},\rho} = (\mathcal{T}_\mathcal{M}, \mathcal{A}_\rho)$.

Model Consistency. The most fundamental service is to check whether $\mathcal{M}$ is consistent, that is, supports the empty trace fragment (in turn witnessesing that it supports at least one full trace). This directly reduces to check whether $\mathcal{T}_\mathcal{M}$ is satisfiable.

Activity Executability. An OCBC model may be consistent, but including so-called *dead activities* [25], i.e., activities that cannot be executed at all. We can show whether an activity A in $\mathcal{M}$ can be executed by verifying whether such an activity is not logically implied to be empty in the corresponding TBox, i.e., $\mathcal{T}_\mathcal{M} \not\models A \sqsubseteq \bot$.

Implied Properties. Let α be a model property expressible in $T_{US}\mathcal{ALCQI}$. We can check whether $\mathcal{M} \models \alpha$ by checking whether $\mathcal{K}_{\mathcal{M},\rho} \models \alpha$. E.g., (3) is a property implied by $\mathcal{M}$. The presented encoding of OCBC into $T_{US}\mathcal{ALCQI}$ allows us to use its reasoning capabilities to detect so-called *hidden constraints* [24], i.e., constraints that are implicitly present in $\mathcal{M}$ even though they are not shown graphically.

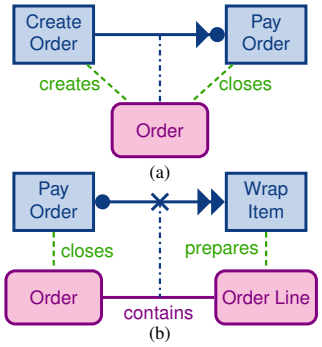


Fig. 7: Implied (a) and non-implied (b) constraints by the OCBC model of Fig. 2

Example 4. Consider again the OCBC model of Fig. 2, and the two constraints in Fig. 7, where Fig. 7a captures that an order can be paid only if it has been created before, and Fig. 7b that no order line of an order can be wrapped after that order is paid. It is easy to verify that the former constraint is indeed implied, while the latter constraint it is not. While it is true that once an order is paid no further items can be picked for it, already picked order lines may still need to be wrapped.

Execution Trace Compliance. This amounts to check whether a trace fragment ρ satisfies the constraints in $\mathcal{M}$. Since ρ is a trace fragment, we require that no explicit violation is contained in ρ and that ρ can be 'completed' into a fully specified, infinite trace that satisfies $\mathcal{M}$. This corresponds to the notion of *conditional compliance* recently introduced in [15]. In our setting, this amounts to check whether the ABox $\mathcal{A}_\rho$ encoding ρ is satisfiable w.r.t. the TBox $\mathcal{T}_\mathcal{M}$, i.e., whether the KB $\mathcal{K}_{\mathcal{M},\rho}$ is satisfiable.

Complexity considerations. Notice that, KB satisfiability and logical implication are mutually reducible in $\mathcal{ALCQI}$ [6] (and thus in $T_{US}\mathcal{ALCQI}$) and these reasoning problems over $T_{US}\mathcal{ALCQI}$ are ExpTime-complete [27, 18], which establishes an ExpTime upper bound for verifying properties of OCBC models. The need to use $\mathcal{ALCQI}$ as the base DL is due to co-reference constraints over relationships, which requires the power of qualified existential ($\exists R.C$) and its dual. If we renounce such constraints (i.e., only consider OCBC constraints co-referring on classes), we could use a temporalized version of a *DL-Lite* dialect. In particular, the temporal *DL-Lite* fragment T_{US}*DL-Lite*$_{bool}^{(\mathcal{HN})}$, showed to be PSpace-complete in [8], is able to capture OCBC models with the exception of co-reference constraints over relationships while, at the level

of the data model, $T_{\mathcal{US}}$DL-Lite$_{bool}^{(\mathcal{HN})}$ captures the main constructs of UML—with the exception of ISA between relationships and n-ary relationships (cf. [4, 7] for details).

6 Conclusions

We presented the first, complete formalization of *object-centric behavioral constraints* (OCBC): a new approach to business process modeling where data models and declarative constraints over activities are seamlessly integrated. Our approach comes with a logic-based semantics for OCBC in terms of an encoding into the temporal DL $T_{\mathcal{US}}\mathcal{ALCQI}$. This unambiguously defines the meaning of OCBC models, and lays the foundations for reasoning over them, allowing us to understand the (decidability and) complexity boundaries of reasoning tasks over OCBC models. $T_{\mathcal{US}}\mathcal{ALCQI}$ interprets time as a linear, infinite structure, which contrasts with the finite-trace semantics adopted in other declarative process modeling languages such as Declare. The study of temporal description logics with finite-time semantics is rather novel [9], and may constitute the basis for reasoning over OCBC models on finite traces.

We have considered here standard data models to capture the structural aspects of OCBC. Variants of OCBC with non-conventional temporalized cardinality constraints over relationships have been used [21, 22]. We intend to study whether such constraints may impact on the decidability and complexity of reasoning over OCBC models.

In our research agenda, we are interested not only in design-time reasoning of OCBC models, but also in enactment, monitoring, and runtime verification. This poses two major challenges. On the one hand, a monitored trace has to be considered under a "partially closed" semantics, that is, by interpreting it as a complete record of what happened so far, while missing information about the future. On the other hand, a more fine-grained analysis, in the style of [23], regarding if and how a monitored trace conforms to an OCBC model is needed. We intend to attack this problem by combining finite and infinite reasoning over a partially closed knowledge base.

Acknowledgments. This research has been partially supported by the UNIBZ CRC projects PWORM and REKAP.

References

1. van der Aalst, W., Pesic, M., Schonenberg, H.: Declarative Workflows: Balancing Between Flexibility and Support. Computer Science–Research and Development **23**(2), 99–113 (2009)
2. van der Aalst, W.M.P.: Process Mining: Data Science in Action. Springer (2016)
3. van der Aalst, W.M.P., Li, G., Montali, M.: Object-Centric Behavioral Constraints. CoRR Technical Report, CoRR (2017), `http://arxiv.org/abs/1703.05740`
4. Artale, A., Calvanese, D., Kontchakov, R., Ryzhikov, V., Zakharyaschev, M.: Reasoning over extended ER models. In: Proc. of the 26th Int. Conf. on Conceptual Modeling (ER). LNCS, vol. 4801, pp. 277–292. Springer (2007)
5. Artale, A., Parent, C., Spaccapietra, S.: Evolving objects in temporal information systems. Annals of Mathematics and Artificial Intelligence **50**(1–2), 5–38 (2007)
6. Artale, A., Calvanese, D., Kontchakov, R., Zakharyaschev, M.: The DL-Lite family and relations. JAIR **36**, 1–69 (2009)

7. Artale, A., Kontchakov, R., Ryzhikov, V., Zakharyaschev, M.: Complexity of reasoning over temporal data models. In: Proc. of the 29th Int. Conf. on Conceptual Modeling (ER). LNCS, vol. 4801, pp. 277–292. Springer (2010)
8. Artale, A., Kontchakov, R., Ryzhikov, V., Zakharyaschev, M.: A cookbook for temporal conceptual data modeling with description logics. ACM Transactivity on Computational Logic (TOCL) **15**(3) (2014)
9. Artale, A., Mazzullo, A., Ozaki, A.: Do you need infinite time? In: In Proc. of the 28th International Joint Conference on Artificial Intelligence (IJCAI) (2019), to appear
10. Bagheri Hariri, B., Calvanese, D., Montali, M., De Giacomo, G., De Masellis, R., Felli, P.: Description logic Knowledge and Action Bases. JAIR **46** (2013)
11. Berardi, D., Calvanese, D., De Giacomo, G.: Reasoning on UML class diagrams. Artificial Intelligence Journal **168**(1–2), 70–118 (2005)
12. Bhattacharya, K., Gerede, C., Hull, R., Liu, R., Su, J.: Towards Formal Analysis of Artifact-Centric Business Process Models. In: Proc. of the 11th Int. Enterprise Distributed Object Computing Conf. (EDOC). LNCS, vol. 4714, pp. 288–304. Springer (2007)
13. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Rosati, R.: Tractable reasoning and efficient query answering in description logics: The *DL-Lite* family. Journal of Automated Reasoning **39**(3), 385–429 (2007)
14. Calvanese, D., De Giacomo, G., Montali, M.: Foundations of data-aware process analysis: A database theory perspective. In: Proc. of 32nd PODS. ACM (2013)
15. Chesani, F., De Masellis, R., Di Francescomarino, C., Ghidini, C., Mello, P., Montali, M., Tessaris, S.: Compliance in business processes with incomplete information and time constraints: a general framework based on abductive reasoning. Fundamenta Informaticae **159**(3), 1–37 (2018)
16. Cohn, D., Hull, R.: Business Artifacts: A Data-centric Approach to Modeling Business Operations and Processes. IEEE Data Engineering Bulletin **32**(3), 3–9 (2009)
17. Franconi, E., Mosca, A., Solomakhin, D.: ORM2: formalisation and encoding in OWL2. In: Proc. of Int. Workshop on Fact-Oriented Modeling (ORM). pp. 368–378 (2012)
18. Gabbay, D., Kurucz, A., Wolter, F., Zakharyaschev, M.: Many-dimensional modal logics: theory and applications. Studies in Logic. Elsevier (2003)
19. Gonzalez, P., Griesmayer, A., Lomuscio, A.: Verification of GSM-Based Artifact-Centric Systems by Predicate Abstractivity. In: Proc. of the 13th Int. Conf. on Service-Oriented Computing (ICSOC). LNCS, vol. 9435, pp. 253–268. Springer (2015)
20. Levesque, H.J.: Foundations of a functional approach to knowledge representation. Artificial Intelligence Journal **23**, 155–212 (1984)
21. Li, G., de Carvalho, R., van der Aalst, W.: Automatic discovery of object-centric behavioral constraint models. In: Int. conf. on Business Information Processing (BIS17) (2017)
22. Li, G., de Carvalho, R., de Murillas, E., van der Aalst, W.: Extracting object-centric event logs to support process mining on databases. In: CAISE Forum. Springer (2108)
23. Maggi, F.M., Westergaard, M., Montali, M., van der Aalst, W.M.P.: Runtime verification of LTL-based declarative process models. In: Proc. of the 2nd International Conference on Runtime Verification (RV). LNCS, vol. 7186, pp. 131–146. Springer (2011)
24. Montali, M., Pesic, M., van der Aalst, W.M.P., Chesani, F., Mello, P., Storari, S.: Declarative specification and verification of service choreographies. ACM Trans. TWEB **4**(1) (2010)
25. Pesic, M., Schonenberg, H., van der Aalst, W.M.: DECLARE: Full support for loosely-structured processes. In: Proc. of the Eleventh IEEE Int. Enterprise Distributed Object Computing Conference (EDOC'07). pp. 287–298. IEEE Computer Society (2007)
26. Vianu, V.: Automatic verification of database-driven systems: a new frontier. In: Proc. of the 12th Int. Conf. on Database Theory (ICDT). pp. 1–13 (2009)
27. Wolter, F., Zakharyaschev, M.: Temporalizing description logics. In: Frontiers of Combining Systems, pp. 379 – 401. Research Studies Press-Wiley (2000)