# Object-Centric Behavioral Constraints: Integrating Data and Declarative Process Modelling

Wil van der Aalst[1], Alessandro Artale[2], Marco Montali[2] and Simone Tritini[2]

[1] Eindhoven University of Technology, P.O. Box 513, 5600 MB, Eindhoven, The Netherlands.
w.m.p.v.d.aalst@tue.nl
[2] Free University of Bozen-Bolzano, Piazza Domenicani 3, I-39100, Bolzano, Italy.
surname@inf.unibz.it

## 1 Introduction

Despite the plethora of notations available to model business processes, process modelers struggle to capture real-life processes in meaningful descriptions. Notations range from formal languages like Petri nets, automata and process algebras to dedicated modeling languages like Business Process Model and Notation (BPMN), Event-driven Process Chains (EPC) and UML activity diagrams. However, all such mainstream modeling languages suffer from two main issues. First, 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. Second, it is *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.

Numerous practical applications of process mining [2] clearly show that there is a mismatch between process models and the data in real enterprise systems from vendors such as SAP (S/4HANA), Microsoft (Dynamics 365), Oracle (E-Business Suite), and Salesforce (CRM). Even though a clear process instance notion is missing in such systems, mainstream business process modeling notations can only describe the *lifecycle of one type of process instance* at a time. To overcome such critical issue and provide a unified representation of process and data-related constraints, the *Object-Centric Behavioral Constraint* (OCBC) model has been recently devised [10]. OCBC combines ideas from declarative, constraint-based languages like *Declare* [12, 11], and from data/object modeling techniques (such as ER, UML, or ORM). Cardinality constraints are used as a *unifying mechanism* to tackle data and behavioral dependencies, as well as their interplay. Unlike existing approaches for process modeling, instances are not considered in isolation and cardinality constraints in the data/object model are taken into account.

The driving modelling assumption underlying our proposal is that a *process* and its composing *actions* are identified by a *unique* instance (a sort of process id). By *typing* the process id with (possibly) different actions at (possibly) different points in time we

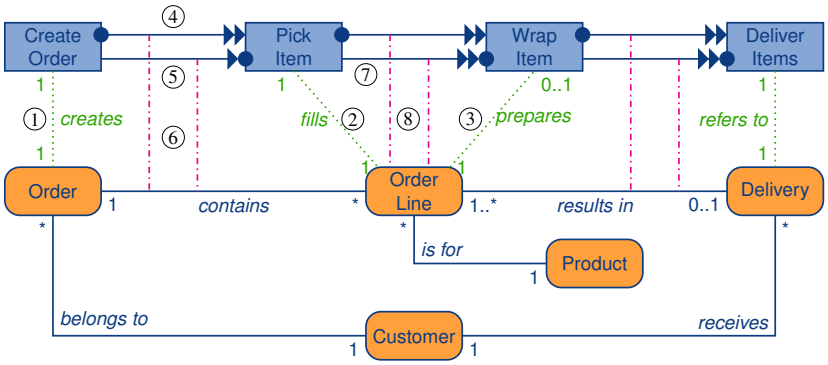


**Fig. 1.** A small *Object-Centric Behavioral Constraint* (OCBC) model.

model the different actions taking place in a given process. The action flows in a process is constrained using LTL-like axioms. To capture objects manipulated by actions we use relations whose domain are the actions and the range the object classes. Constraints are added to express possible co-references between objects manipulated by different actions. The following example shows the main ingredients of our proposal.

*Example 1.* Fig. 1 shows an OCBC model for a process composed by four activities (`CreateOrder`, `PickItem`, `WrapItem` and `DeliverItems`) and five object classes (`Order`, `OrderLine`, `Delivery`, `Product` and `Customer`). The top part describes the ordering of activities and the bottom part the structuring of objects relevant for the process. The lower part can be read as a standard UML class diagram (e.g., an `Order` contains zero or more `OrderLines` while each `OrderLine` corresponds to precisely one `Order`, each `OrderLine` refers to one `Product`, each `Order` refers to one `Customer`, etc.). The top part shows behavioral constraints between the activities of the process and the middle part relates activities and data. To introduce the main ideas underlying our proposal, we informally describe the constructs highlighted in Fig. 1. ① There is a one-to-one correspondence between a `CreateOrder` activity and an `Order`, i.e., the execution of a `CreateOrder` activity creates to a unique `Order`. ② Every execution of the `PickItem` activity refers to a unique `OrderLine` and each `OrderLine` can be picked exactly once. ③ Every execution of the `WrapItem` activity refers to an `OrderLine` and each `OrderLine` can be wrapped at most once. ④ Each `CreateOrder` activity is followed by one or more `PickItem` activities related to the same order. ⑤ Each `PickItem` activity is preceded by precisely one `CreateOrder` activity. ⑥ The two co-reference constraints (dash dotted lines) impose that when we create an order instance it will contain an order line eventually associated to a `PickItem` activity and, viceversa, to each order line associated to a `PickItem` activity corresponds an order as created by the `CreateOrder` activity. ⑦ Each `WrapItem` activity is preceded by one or more `PickItem` activities. ⑧ Two co-reference constraints (dash dotted lines) impose that when a `PickItem` activity `fills` an order line instance it will be the same as the one prepared by a `WrapItem` activity and, viceversa, an `OrderLine` prepared by a `WrapItem` must coincide with that one filled by a `PickItem` activity.

A possible model of the OCBC model of Fig 1 is the following (we abbreviate names of activities and classes with their initials):

$\texttt{CO}(p_1,t_0), \texttt{PI}(p_1,t_1), \texttt{PI}(p_1,t_2), \texttt{WI}(p_1,t_3), \texttt{WI}(p_1,t_4), \texttt{PI}(p_1,t_5), \texttt{WI}(p_1,t_6), \texttt{DI}(p_1,t_7), \texttt{DI}(p_1,t_8),$
$\texttt{creates}(p_1,o_1,t_0), \texttt{contains}(o_1,ol_1,t_1), \texttt{fills}(p_1,ol_1,t_1), \texttt{contains}(o_1,ol_2,t_2), \texttt{fills}(p_1,ol_2,t_2),$
$\texttt{prepares}(p_1,ol_1,t_3), \texttt{prepares}(p_1,ol_2,t_4), \texttt{contains}(o_1,ol_3,t_5), \texttt{fills}(p_1,ol_3,t_5), \texttt{prepares}(p_1,ol_3,t_6),$
$\texttt{results in}(ol_1,d_1,t_7), \texttt{results in}(ol_2,d_1,t_7), \texttt{refers to}(p1_1,d_1,t_7),$
$\texttt{results in}(ol_3,d_2,t_8), \texttt{refers to}(p1_1,d_2,t_8).$

Note that the `DeliverItems` activity can deliver one or more `OrderLines` and that the `OrderLines` contained in an `Order` can be scattered over multiple `Deliveries`.

The process described in the previous 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 constraints and relations in the object class model influence the allowed behavior of activities and vice-versa. In the above example, interesting implicit constraints emerge from the interplay of the temporal constraints between activities and the cardinality constraints on activity-class and class-class relationships. For example, the temporal ordering of the activities logically implies that each `Order` will eventually contain at least one `OrderLine`.

In this paper, we focus on the formal semantics of the OCBC model, so as to unambiguously define the *meaning* of OCBC constraints, and in particular how the contribution of the data and that of the temporal constraints among activities are intertwined. To do so, we employ *first-order logic over the flow of time*, i.e., first-order logic equipped with a special sort that represents time points and the usual $<$ rigid binary relation. We then encode the resulting logical theory using *temporal description logics*, consequently paving the way towards automated reasoning over OCBC models and the identification of the corresponding boundaries of decidability and tractability.

## 2 Temporal Description Logics

In this paper we use the temporal description logic $\mathcal{T}_{\mathcal{US}}$*DL-Lite*$_{bool}^{(\mathcal{HN})}$ [6] to capture in a uniform formalism both the activities and their attached data. It is known from temporal logic [9] that all the temporal operators used in Linear Temporal Logic (LTL) can be expressed via $\mathcal{S}$ 'since' and $\mathcal{U}$ 'until'. $\mathcal{T}_{\mathcal{US}}$*DL-Lite*$_{bool}^{(\mathcal{HN})}$ is one of the most expressive and still decidable temporal description logics which uses these two temporal operators. The language contains *concept names* $CN_0, CN_1, \ldots$, *flexible role names* $P_0, P_1, \ldots$, and *rigid role names* $G_0, G_1, \ldots$. *Role names* $S$, *roles* $R$, *basic concepts* $B$ and *(temporal) concepts* $C$ are given by the following grammar:

$$\begin{array}{lll} S & ::= & P_i \mid G_i \qquad \text{and} \qquad R ::= S \mid S^- \\ B & ::= & \bot \mid CN_i \mid \geq q\, R \\ C & ::= & B \mid \neg C \mid C_1 \sqcap C_2 \mid C_1\, \mathcal{U}\, C_2 \mid C_1\, \mathcal{S}\, C_2 \end{array}$$

where $S^-$ denotes the *inverse* of the role $S$ and $q$ is a positive integer. We use the standard abbreviations: $C_1 \sqcup C_2 = \neg(\neg C_1 \sqcap \neg C_2), \top = \neg\bot, \exists R = (\geq 1\, R), \leq q\, R =$

$\neg(\geq q+1\, R)$. A $T_{\mathcal{US}}$*DL-Lite*$^{(\mathcal{HN})}_{bool}$ *TBox*, $\mathcal{T}$, is a finite set of *concept inclusion* axioms of the form $C_1 \sqsubseteq C_2$, and of *role inclusion* axioms of the form $R_1 \sqsubseteq R_2$.

A *temporal interpretation* is a structure of the form $\mathcal{I} = ((\mathbb{Z}, <), \Delta^{\mathcal{I}}, \{\cdot^{\mathcal{I}(n)} \mid n \in \mathbb{Z}\})$, 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 $n \in \mathbb{Z}$: $\mathcal{I}(n) = \big(\Delta^{\mathcal{I}}, CN_0^{\mathcal{I}(n)}, \dots, P_0^{\mathcal{I}(n)}, \dots, G_0^{\mathcal{I}}, \dots\big)$, assigning to each concept name $CN_i$ a subset $CN_i^{\mathcal{I}(n)} \subseteq \Delta^{\mathcal{I}}$, to each flexible role name $P_i$ a binary relation $P_i^{\mathcal{I}(n)} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$, while the interpretations $G_i^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ of rigid role names are the same for all $n \in \mathbb{Z}$. The role and concept constructs are interpreted as follows, where $C^{\mathcal{I}(n)}$ ($R^{\mathcal{I}(n)}$) denotes the extension of $C$ ($R$) at time $n \in \mathbb{Z}$:

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

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 $\mathcal{U}$ (as well as the $\Box$ and $\Diamond$ operators to be defined below) are 'strict' in the sense that their semantics does not include the current moment of time. The non-strict operators, which include the current moment, are obviously definable in terms of the strict ones. Concept and role inclusion axioms are interpreted in $\mathcal{I}$ *globally*:

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

with the following restriction on the interaction between role inclusions and cardinalities: no role $R$ can occur in $\mathcal{T}$ in both a role inclusion of the form $R' \sqsubseteq R$ and a number restriction $\geq q\, R$ or $\geq q\, R^-$ with $q \geq 2$.

We call $\mathcal{I}$ a *model* of a TBox $\mathcal{T}$ and write $\mathcal{I} \models \mathcal{T}$ if $\mathcal{I}$ satisfies all inclusions in $\mathcal{T}$. A TBox $\mathcal{T}$ is *satisfiable* if it has a model. A concept $C$ (role $R$) is *satisfiable* with respect to $\mathcal{T}$ if there are a model $\mathcal{I}$ of $\mathcal{T}$ 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 TBox satisfiability.

We now recall how to capture the other LTL operators (used in this paper) via the $\mathcal{U}$ and $\mathcal{S}$ operators. The operators $\Diamond_F$ and $\Diamond_P$ ('sometime in the future/past') can be expressed as $\Diamond_F C = \top \mathcal{U} C$ and $\Diamond_P C = \top \mathcal{S} C$; the operators $\Box_F$ ('always in the future') and $\Box_P$ ('always in the past') are defined as dual to $\Diamond_F$ and $\Diamond_P$: $\Box_F C = \neg\Diamond_F\neg C$ and $\Box_P C = \neg\Diamond_P\neg C$. The nclusion $C \sqsubseteq \circledast C$ captures rigid concepts by using the 'always' operator $\circledast$, that can be expressed as $\circledast C = \Box_F\Box_P C$, while the dual operator 'sometime' is defined as usual: $\circledcirc C = \neg \circledast \neg C$. 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 $\bigcirc_P C = \bot\, \mathcal{S}\, C$.

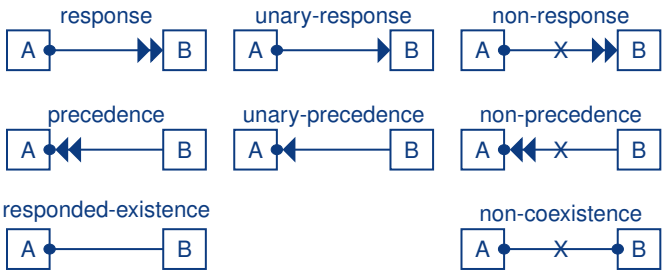


**Fig. 2.** Types of BCM constraints.

Reasoning in $T_{\mathcal{US}}$*DL-Lite*$_{bool}^{(\mathcal{HN})}$ w.r.t. to a TBox is a decidable problem which has been proven to be PSpace-complete in [6], i.e., the complexity of reasoning in LTL.

## 3 The OCBC Model

In this section we present the main results of this paper, a formalization of what we call the OCBC model, i.e., a model where Object Classes and Behavioral Constraints are both present. In particular, the OCBC model simultaneously accounts for: *(i) control-flow constraints*, captured as declarative temporal dependencies between activities, taking inspiration from the Declare language [1]; *(ii) data dependencies*, captured using standard data models that account for classes, relationships, and cardinality constraints; *(iii) mutual relationships between activities and classes*, so as to link the execution steps in the process with the data objects they manipulate; *(iv) coreference constraints* between instances of the data model associated to activities via the above mentioned relationships linking activities to data elements.

We proceed as follows: we first show how behavioral constraints between activities (the so called Behavioral Constraints Model, BCM) can be formalized in Linear Temporal Logic (LTL). As for the data model, we rely on well-established foundational results showing how standard data models can be suitably formalized in FOL and, in turn, encoded into suitable DLs for reasoning [7, 3, 8]. We finally enrich the formalization by accounting for activity-class relationships, and coreference constraints.

### 3.1 The Behavioral Constraints Model BCM

The BCM captures a set of declarative temporal constraints between (pairs of) activities, taking inspiration from the relation and negation Declare patterns [1]. Fig. 2 graphically renders the repertoire of behavioral constraints considered in this paper (also briefly sketching their relationship with Declare), while their textual representation is defined next.

**Definition 1 (BCM).** *A* BCM *is a triple:* $(\mathcal{U}_A, \mathcal{U}_{BC}, \Sigma_{BC})$*, where:*

– $\mathcal{U}_A$ *is the universe of the activities, denoted with capital letters* $A_1, A_2, \ldots$*;*

| $\texttt{Response}(A, B)$ | |
|---|---|
| $A \sqsubseteq \Diamond_F B$ | If A executes then B must be executed afterwards. |
| $\texttt{Unary-Response}(A, B)$ | |
| $A \sqsubseteq \neg B\, \mathcal{U}\, (B \sqcap \Box_F \neg B)$ | If A executes then B must be executed exactly once afterwards. |
| $\texttt{Precedence}(A, B)$ | |
| $A \sqsubseteq \Diamond_P B$ | If A executes then B must have been executed before. |
| $\texttt{Unary-Precedence}(A, B)$ | |
| $A \sqsubseteq \neg B\, \mathcal{S}\, (B \sqcap \Box_P \neg B)$ | If A executes then B must have been executed exactly once before. |
| $\texttt{Responded-Existence}(A, B)$ | |
| $A \sqsubseteq \Diamond\!\!\!\!-\, B$ | If A executes then B must be executed either before or afterwards. |
| $\texttt{Non-Response}(A, B)$ | |
| $A \sqsubseteq \Box_F \neg B$ | If A executes then B will never be executed afterwards. |
| $\texttt{Non-Precedence}(A, B)$ | |
| $A \sqsubseteq \Box_P \neg B$ | If A executes then B was never executed before. |
| $\texttt{Non-Coexistence}(A, B)$ | |
| $A \sqsubseteq \circledast \neg B$ | A and B cannot be both executed. |

**Fig. 3.** Semantics of control-flow constraints, where A and B are activities

- $\mathcal{U}_{BC}$ *is the universe of the* behavioral constraints *that can be expressed between activities,* $\mathcal{U}_{BC} = \{$*responded-existence, response, unary-response, precedence, unary-precedence, non-response, non-precedence, non-coexistence*$\}$*, as shown in Fig. 2, where each* $bc \in \mathcal{U}_{BC}$ *is a binary relation over activities, i.e.,* $bc \subseteq \mathcal{U}_A \times \mathcal{U}_A$*;*
- $\Sigma_{BC}$ *is the set of* control-flow constraints *of the form* $bc(A_1, A_2)$*, where* $bc \in \mathcal{U}_{BC}$ *and* $A_1, A_2 \in \mathcal{U}_A$*.*

When defining later on the OCBC model we will consider the set $\mathcal{U}^+_{BC}$ of *positive* behavioral constraints, containing *response*, *unary-response*, *precedence*, *unary-precedence*, and *responded-existence*. The formal semantics of control-flow constraints is captured via $T_{\mathcal{US}}$*DL-Lite*$^{(\mathcal{HN})}_{bool}$ concept inclusion axioms (LTL-like formulas since roles are absent), as shown in Fig. 3 together with their intuitive meaning.

*Example 2.* Fig. 4 shows an example of a BCM describing the process flow of buying a ticket and its correlated activities. The arrow between *SelectFlight* and *Payment* represents an *unary-precedence* constraint, thus when *Payment* is executed there must be a single execution of *SelectFlight* in the past. In case we *Cancel Ticket* then sometime in the past we should have done the *Payment*. We provide also examples of negative behavioral constraints, e.g., when a *Refund* is executed then *Check-In* is never executed, i.e., if we cancel and then refund the ticket we can not do the check-in anymore. The corresponding $T_{\mathcal{US}}$*DL-Lite*$^{\mathcal{N}}_{bool}$ axioms capturing the BCM of this example are:

$$\begin{aligned}
\textit{Payment} &\sqsubseteq \neg \textit{SelectFlight}\, \mathcal{S}\, (\textit{SelectFlight} \sqcap \Box_P \neg \textit{SelectFlight}),\\
\textit{Payment} &\sqsubseteq \neg \textit{PrintTicket}\, \mathcal{U}\, (\textit{PrintTicket} \sqcap \Box_F \neg \textit{PrintTicket}),\\
\textit{CancelTicket} &\sqsubseteq \Diamond_P \textit{Payment}, \quad \textit{CancelTicket} \sqsubseteq \Diamond_F \textit{Refund},\\
\textit{Refund} &\sqsubseteq \circledast \neg \textit{Check-In}.
\end{aligned}$$

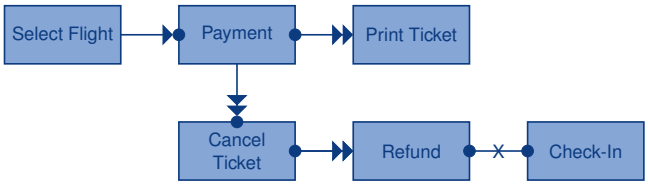


**Fig. 4.** BCM example capturing the fragment of a ticket purchase process

### 3.2 The Class Model - ClaM

We assume that data used by the activities conform to the ClaM data model. In this paper we consider a data model with basic constructs. For simplicity, we define here ClaM as a simplified version of UML, with object classes that can be organized along ISA hierarchies, binary relationships between object classes and cardinalities expressing participation constraints of object classes in relationships. More formally we have the following:

**Definition 2 (ClaM Syntax).** *A conceptual schema $\Sigma$ in the Class Model, ClaM, is a tuple $\Sigma = (\mathcal{U}_C, \mathcal{U}_R, \tau, \#_{src}, \#_{tar}, \mathsf{ISA})$, 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 $R_1, R_2, \ldots$;*
- *$\tau : \mathcal{U}_R \rightarrow \mathcal{U}_C \times \mathcal{U}_C$ 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;*
- *$\#_{dom} : \mathcal{U}_R \times \mathcal{U}_C \not\rightarrow \mathbb{N} \times (\mathbb{N} \cup \{\infty\})$ is a* partial *function defining* cardinality constraints *of the domain of a relationship. The value $\#_{dom}(R, O)$ is defined only if $\tau(R) = (O, O_1)$;*
- *$\#_{ran} : \mathcal{U}_R \times \mathcal{U}_C \not\rightarrow \mathbb{N} \times (\mathbb{N} \cup \{\infty\})$ is a* partial *function defining* cardinality constraints *of the range of a relationship. The value $\#_{ran}(R, O)$ 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 $C_1$.*

As for the formal set-theoretic semantics of ClaM and its translation to description logics we refer to [7, 3]. In particular, cardinality constraints are interpeted as the number of times a given instance of the involved object class participates in the given relationships, while ISA is interpreted as sub-setting. To better clarify the elements of ClaM we show the following example.

*Example 3.* We consider the example shown in Fig. 5, modelling a process flow (upper part) and its associated data modeled via a ClaM diagram. Concerning the ClaM

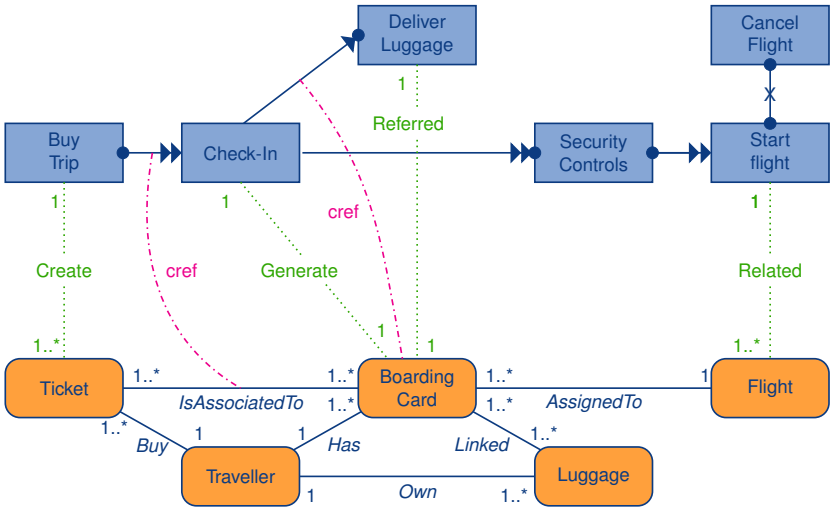


**Fig. 5.** The OCBCM example of an airplane trip scenario.

diagram we have that:

$$\mathcal{U}_C = \{\textit{Ticket, Traveller, Boarding-Card, Luggage, Flight}\};$$
$$\mathcal{U}_R = \{\textit{IsAssociatedTo, Buy, Has, Own, Linked, AssignedTo}\};$$
$$\tau(\textit{Own}) = (\textit{Traveller,Luggage}), \ldots$$
$$\#_{dom}(\textit{Own,Traveller}) = (1,\infty);\ \ \#_{ran}(\textit{Own,Luggage}) = (1,1); \ldots$$

Note that cardinalities are depicted in the diagram in UML style.

### 3.3 The Object Centric Behavioral Constraint Model - OCBC

The Object Centric Behavioral Constraint Model (OCBC) combines the behavioral constraints model BCM capturing the process flow (as presented in Section 3.1) with the object classes represented by the ClaM data model (as presented in Section 3.2). The original proposal in the OCBC model is the way activities and data are related to each other and the formal underpinning of the model. We now present in details the syntax of an OCBC model.

**Definition 3 (The OCBC syntax).** *An OCBC model is a tuple:*

$$(\textit{BCM}, \textit{ClaM}, \mathcal{U}_{R_{AC}}, \tau_{R_{AC}}, \#_{src}, \#_{tar}, \mathcal{U}_{crel})$$

*where:*

- *BCM is a behavioral constraint model as in Definition 1;*
- *ClaM is a conceptual schema as in Definition 2;*
- $\mathcal{U}_{R_{AC}}$ *is the universe of* activity-object relationships *being a set of binary relationships;*

- $\tau_{R_{AC}} : \mathcal{U}_{R_{AC}} \rightarrow \mathcal{U}_A \times \mathcal{U}_C$ *is a* total *function associating a signature to each activity-object relationship. If* $\tau_{R_{AC}}(R) = (A, O)$ *then* $A \in \mathcal{U}_A$ *is the* source *and* $O \in \mathcal{U}_C$ *the* target *of the relationship;*
- $\#_{src} : \mathcal{U}_{R_{AC}} \times \mathcal{U}_A \nrightarrow \mathbb{N} \times (\mathbb{N} \cup \{\infty\})$ *is a* partial *function defining* cardinality constraints *on activities, i.e., constraints on the participation of activities in activity-object relationships. The value* $\#_{src}(R, A)$ *is defined only if* $\tau_{R_{AC}}(R) = (A, O)$*;*
- $\#_{tar} : \mathcal{U}_{R_{AC}} \times \mathcal{U}_C \nrightarrow \mathbb{N} \times (\mathbb{N} \cup \{\infty\})$ *is a* partial *function defining* cardinality constraints *on object classes, i.e., constraints on the participation of object classes in activity-object relationships. The value* $\#_{tar}(R, O)$ *is defined only if* $\tau_{R_{AC}}(R) = (A, O)$*;*
- $\mathcal{U}_{coref}$ *is the universe of* coreference constraints *being a set of functions, i.e.,* $\mathcal{U}_{coref} = \{cr \mid cr : \Sigma_{BC} \times \mathcal{U}_{R_{AC}} \times \mathcal{U}_{R_{AC}} \rightarrow \mathcal{U}_C \cup \mathcal{U}_R\}$.

To better understand the expressive power of the OCBC modelling language we discuss the scenario of an airplane travel.

*Example 4.* Fig. 5 shows how an OCBC diagram captures an airplane trip scenario. The activities that belong to the process flow and modeled as a BCM diagram are depicted in the upper part of the figure. Then, in the lower part of the figure, we have the ClaM data model that captures the data manipulated by the activities of the process flow. The set $\mathcal{U}_{R_{AC}}$ of the activity-object relationships is the following set of binary relationships:

$$\mathcal{U}_{R_{AC}} = \{\textit{Create}, \textit{Generate}, \textit{Referred}, \textit{Receive}, \textit{Related}\},$$

connecting an activity with the objects manipulated as an effect of executing the activity itself. For example, the activity *BuyTrip* creates an instance of the object class *Ticket* when it is executed. Cardinality constraints can also be added to activity-object relationships to specify participation constraints either on the activity side or on the object class side. For example, at any point in the time, an execution of *Check-in* creates exactly one *BoardingCard* while each *BoardingCard* corresponds to exactly one *Check-in* action. Thus, $\#_{src}(\textit{Generate}, \textit{Check-in}) = \#_{tar}(\textit{Generate}, \textit{BoardingCard}) = (1, 1)$. On the other hand, when we execute *BuyTrip* we can buy one or more tickets while a *Ticket* is associated to a single *BuyTrip* action. The *coreference constraints* (the dashed-dotted lines denotes as *cref* in Fig. 5) specify constraints on how objects connected to different activities can be shared. For example, the *BoardingCard* generated by a *Check-in* is the same used to deliver the luggage. This particular coreference constraint is specified as:

$$\textit{cref}(\texttt{Unary-Precedence}(\textit{DeliverLuggage}, \textit{Check-in}), \textit{Referred,Generate}) = \textit{BoardingCard},$$

while the other coreference constraints in Fig. 5 is expressed as:

$$\textit{cref}(\texttt{Response}(\textit{ButTrip}, \textit{Check-in}), \textit{Create,Generate}) = \textit{IsAssociatedTo}.$$

In the next section we will present the semantics of OCBC models and we better clarify the two kinds of coreference constraints that can be involved in an OCBC model.

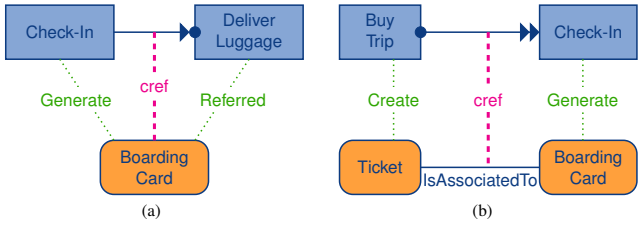


**Fig. 6.** The two kinds of coreference: (a) over an object class, (b) over a relationship.

## 4 The OCBC Formal Semantics

This section presents the semantics for OCBC models. In this respect, our effort is to reconcile the process flow semantics with the data model semantics associating to both worlds an FOL formalization and the corresponding temporal DL axioms.

To capture the temporal nature of the whole framework we use a two sorted FOL with a sort dedicated to the time dimension. Thus, in the following, the variable $t$ will denote a time point to be interpreted as an integer number. FOL formulas map activities and object classes to binary predicates while activity-object relationships and relationships of the data model to ternary relations. More formally,

$$\begin{aligned} A \in \mathcal{U}_A \cup \mathcal{U}_C \quad &\text{is mapped in FOL as} \quad A(x,t); \\ R \in \mathcal{U}_{R_{AC}} \cup \mathcal{U}_R \quad &\text{is mapped in FOL as} \quad R(x,y,t). \end{aligned}$$

As for the semantics of a BCM model, we already gave its meaning in Fig. 3 via the temporal DL translation. Concerning the semantic of the ClaM data model, we interpret it along the temporal semantics presented in [4, 5] for temporal data models. In the same papers, a mapping from a (temporal) data model to a temporal DL TBox is presented and used in this paper. It is now crucial to formalize the meaning of *coreference constraints*. We proceed by assigning an FOL translation and then the corresponding temporal DL in the form of a $\mathcal{TUS}$*DL-Lite*$_{bool}^{(\mathcal{HN})}$ TBox extended with temporalized roles, i.e., roles of the form $\Diamond_P R$, $\Diamond_F R$, $\Box_P R$, $\Box_F R$, $\bigcirc_P R$, $\bigcirc_F R$, with the obvious meaning.

All together, we will show how an OCBC model can be captured via a TBox in $\mathcal{TUS}$*DL-Lite*$_{bool}^{(\mathcal{HN})}$ thus resulting in a uniform representation. According to Definition 3, there are two kinds of coreference constraints: the ones that range over object classes and the ones ranging over relationships. We start with the coreference over object classes as illustrated, e.g., in Fig. 6(a).

**Definition 4 (Semantics of coreference constraints over object classes).** *Let* $cr \in \mathcal{U}_{coref}$, $bc \in \mathcal{U}_{BC}^{+}$, $R_1, R_2 \in \mathcal{U}_{R_{AC}}$, $A_1, A_2 \in \mathcal{U}_A$ *and* $O \in \mathcal{U}_C$ *s.t.* $bc(A_1, A_2) \in \Sigma_{BC}$, $\tau(R_1) = (A_1, O)$, $\tau(R_2) = (A_2, O)$ *and* $cr(bc(A_1, A_2), R_1, R_2) = O$. *Then, the following FOL formula (in brackets the corresponding* $\mathcal{TUS}$*DL-Lite*$_{bool}^{(\mathcal{HN})}$ *axioms) captures* domain *and* range *restrictions:*

$$\forall x, y, t. R_1(x, y, t) \rightarrow A_1(x, t) \wedge O(y, t) \qquad (\exists R_1 \sqsubseteq A_1, \quad \exists R_1^- \sqsubseteq O), \qquad (1)$$

$$\forall x, y, t. R_2(x, y, t) \rightarrow A_2(x, t) \wedge O(y, t) \qquad (\exists R_2 \sqsubseteq A_2, \quad \exists R_2^- \sqsubseteq O), \qquad (2)$$

*while the semantics of the coreference is the following (in case bc is the constraint* $\texttt{Response}(A_1, A_2)$*):*

$$\forall x, y, t. R_1(x, y, t) \rightarrow \exists t' > t. R_2(x, y, t') \qquad (R_1 \sqsubseteq \Diamond_F R_2). \tag{3}$$

*Similar formulas hold for other forms of positive behavioral constraints.*

We now consider the coreference over relationships as illustrated, e.g., in Fig. 6(b). In this case we need to consider two object classes in the data model that are related together with a relationship on which the coreference holds. In the DL translation, we need to use a *role chain* constructor with the following meaning: $(R_1 \circ R_2)^{\mathcal{I}(n)} = \{(x, y) \in \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \mid \exists z.(x, z) \in R_1^{\mathcal{I}(n)} \wedge (z, y) \in R_2^{\mathcal{I}(n)}\}$.

**Definition 5 (Semantics of coreference constraints over relationships).** *Let* $cr \in \mathcal{U}_{coref}$*,* $bc \in \mathcal{U}_{BC}^{+}$*,* $R_1, R_2 \in \mathcal{U}_{R_{AC}}$*,* $A_1, A_2 \in \mathcal{U}_A$*,* $O_1, O_2 \in \mathcal{U}_C$ *and* $R \in \mathcal{U}_R$ *a relationships between* $O_1$ *and* $O_2$ *s.t.* $bc(A_1, A_2) \in \Sigma_{BC}$*,* $\tau(R_1) = (A_1, O_1)$*,* $\tau(R_2) = (A_2, O_2)$ *and* $cr(bc(A_1, A_2), R_1, R_2) = R$*. Then, the semantics of* domain *and* range *restrictions is as in Def. 4, while the semantics of the coreference when bc is a future constraint is the following (in case bc is the constraint* $\texttt{Response}(A_1, A_2)$*):*

$$\forall x, y, t. R_1(x, y, t) \rightarrow \exists z, t'. t' > t \wedge R_2(x, z, t') \wedge R(y, z, t') \quad (R_1 \sqsubseteq \Diamond_F(R_2 \circ R^{-})), \tag{4}$$

*and when bc is a past constraint then (in case bc is the constraint* $\texttt{Precedence}(A_1, A_2)$*):*

$$\forall x, y, t. \exists z. R_1(x, z, t) \wedge R(z, y, t) \rightarrow \exists t'. t' < t \wedge R_2(x, y, t') \quad (R_1 \circ R \sqsubseteq \Diamond_P R_2). \tag{5}$$

*Similar formulas hold for other forms of behavioral constraints.*

## 5 Considerations on Reasoning over OCBC Models

The main motivation to provide a mapping to a DL TBox is the possibility offered by DLs to reason over TBoxes. As we observed, the temporal description logic used in this paper, $T_{\mathcal{US}}$*DL-Lite*$_{bool}^{(\mathcal{HN})}$, is decidable and PSpace-complete. $T_{\mathcal{US}}$*DL-Lite*$_{bool}^{(\mathcal{HN})}$ is able to capture BCM diagrams thanks to its temporal capabilities. At the level of data models, $T_{\mathcal{US}}$*DL-Lite*$_{bool}^{(\mathcal{HN})}$ captures the main constructs of UML—with the exception of ISA between relationships and n-ary relationships—adding the possibility to express temporal constraints over both object classes and relationships (see [3, 5] for details).

On the other hand, to fully capture OCBC models we need to go beyond the expressivity of $T_{\mathcal{US}}$*DL-Lite*$_{bool}^{(\mathcal{HN})}$. Indeed, due to coreference constraints we need the expressivity of temporalised roles (see axiom (3)) or role chains (see axioms (4)-(5)). Both constructors can ruin the decidability of the resulting language. So, while reasoning over OCBC models without corefence constraints is a PSpace-complete problem the addition of coreferences makes reasoning an undecidable problem.

One possibility to regain decidability, admitting just corefences over object classes, is to avoid at-most cardinality constraints on activity-object constraints (the undecidability proof in [6] relies on both temporalised roles and on the possibility to represent functional roles). The case with coreference constraints over relationships is more involved and requires further investigations. Indeed, it is well known that role inclusion axioms with role chains on the right-hand side (i.e., axioms of the form $R \sqsubseteq R_1 \circ R_2$) make the logic undecidable. It is to be understood whether the special form of role chains in OCBC models can still encode an undecidable problem.

## References


1. van der Aalst, W.M.P., Pesic, M.: Decserflow: Towards a truly declarative service flow language. In: The Role of Business Processes in Service Oriented Architectures (2006)
2. van der Aalst, W.M.P.: Process Mining: Data Science in Action. Springer (2016)
3. 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'07). LNCS, vol. 4801, pp. 277–292. Springer (2007)
4. Artale, A., Parent, C., Spaccapietra, S.: Evolving objects in temporal information systems. Annals of Mathematics and Artificial Intelligence 50(1–2), 5–38 (2007)
5. 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'10). LNCS, vol. 4801, pp. 277–292. Springer (2010)
6. Artale, A., Kontchakov, R., Ryzhikov, V., Zakharyaschev, M.: A cookbook for temporal conceptual data modelling with description logics. ACM Transaction on Computational Logic (TOCL) 15(3) (2014)
7. Berardi, D., Calvanese, D., De Giacomo, G.: Reasoning on UML class diagrams. Artificial Intelligence 168(1–2), 70–118 (2005)
8. Franconi, E., Mosca, A., Solomakhin, D.: ORM2: formalisation and encoding in OWL2. In: Int. Workshop on Fact-Oriented Modeling (ORM'12). pp. 368–378 (2012)
9. Gabbay, D., Hodkinson, I., Reynolds, M.: Temporal Logic: Mathematical Foundations and Computational Aspects, vol. 1. Oxford University Press (1994)
10. Li, G., Montali, M., van der Aalst, W.M.P.: Object-centric behavioral constraints. Corr technical report, arXiv.org e-Print archive (2017)
11. Montali, M., Pesic, M., van der Aalst, W.M.P., Chesani, F., Mello, P., Storari, S.: Declarative specification and verification of service choreographiess. ACM Transactions on the Web (TWEB) 4(1) (2010)
12. 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)