Ontology Makes Sense
S. Borgo et al. (Eds.)
IOS Press, 2019

doi:10.3233/978-1-61499-955-3-257

Enriching Data Models with Behavioral Constraints

Alessandro Artale, Diego Calvanese, Marco Montali [a] and Wil van der Aalst [b]
[a] *KRDB Research Centre, Free University of Bozen-Bolzano, Italy*
{artale,calvanese,montali}@inf.unibz.it
[b] *Process and Data Science (PADS), RWTH Aachen University, Germany*
wvdaalst@pads.rwth-aachen.de

Abstract. Existing process modelling notations ranging from Petri nets to BPMN have difficulties capturing the essential features of the domain under study. 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. In addition, they essentially rely on the simplifying assumption that each process model focuses on a single, explicitly defined notion of case, representing the type of objects that are separately manipulated when the process is instantiated into actual executions. To overcome this key limitation, *Object-Centric Behavioural Constraints* (OCBC) models were recently proposed as a new notation where data and control-flow are described in a single diagram, and where their interconnection is exploited to elegantly capture real-life processes operating over a complex network of objects. In this paper, we illustrate the essential and distinctive features of the OCBC approach, and contrast OCBC with contemporary, case-centric notations. We then relate the approach to recent developments in the conceptual understanding of processes, events, and their constituents, introducing a series of challenges and points of reflections for the community.

1. Introduction

Despite the plethora of notations available to model business processes, process modellers 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 notion of *case*, representing the type of objects that are separately manipulated when the process is instantiated into actual executions. The discrepancy between this assumption and reality becomes evident when using process mining techniques to reconstruct the real processes based on the available data [1]. Process mining starts from the available data and, unless one is using a Business Process Management (BPM) of Workflow Management (WFM) system for process executions, explicit case information is typically missing. Real enterprise systems from vendors such as SAP (S/4HANA), Microsoft (Dynamics 365), Oracle (E-Business Suite), and

Salesforce (CRM) are database-centric. Process activities can be viewed as updates on the underlying data, and the notion of process instance (i.e., case) does not exist.

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 also tend to model cases in isolation. Moreover, the data perspective is secondary or missing completely. *Object-Centric Behavioural Constraint* (OCBC) models [2] have been recently proposed to address the following interrelated problems:

- Modelling the *interaction between multiple process instances*, specifically when there is a *one-to-many* or *many-to-many* relationship between them. For example, an order may relate to multiple order lines and one delivery may refer to multiple order lines belonging to different orders. How to model activities related to orders, order lines, and deliveries in a single diagram?
- Modelling *both* the data and the control-flow perspective in a *unified* and *integrated* manner. Existing approaches focus on one of the two or resort to the use of multiple diagrams. For example, the relation between class diagrams and activity diagrams in UML is indirect and not visualized.

Note that languages like BPMN allow modellers to attach simple data objects to processes, but the more powerful constructs present in Entity Relationship (ER) models and UML class models cannot be captured in such process models. In particular, complex constraints over data attached to processes (e.g., cardinality constraints) *must* influence the behaviour of the process itself (think to the activities that must generate those data objects). In contemporary languages, neither complex constraints over data nor a way to capture how data can influence processes are reflected at all. Also other mainstream business process modelling notations can only describe the *lifecycle of one type of process instance* at a time, but not the co-evolution of multiple, interacting instances (such as the management of different orders, where the evolution of one order impacts on the possible evolutions of the related orders).

Object-Centric Behavioural Constraint (OCBC) models have been proposed as a modelling language that combines ideas from declarative, constraint-based languages like DECLARE [3], and from structural conceptual modelling languages (such as ER, UML, or ORM) [2]. OCBC allows to describe the temporal interaction between activities in a given process and is able to attach (structured) data to processes in a *unified framework*. In this way, we can capture in a uniform way processes, data, and their interplay.

OCBC models are related to artifact- and data-centric approaches [4, 5, 6]. These approaches also aim to integrate data and processes. However, this is not done in a single diagram representing different types of process instances and their interactions (which are governed by the data). In addition, these approaches usually assume complete knowledge over the data, and require to fully spell out data updates when specifying the activities [7, 8]. The few proposals dealing with artifact-centric models whose structural aspects are interpreted under incomplete information [9] do not come with a fully integrated, declarative semantics, but follow instead the Levesque functional approach [10] to separate the evolution of the system from the inspection of structural knowledge in each state. The

semantics of an OCBC model, instead, can be fully characterised by resorting to *first-order temporal logic*, where the temporal dimension capturing the dynamics of the process interacts with the structural dimension of objects and their mutual relationships [11].

In this paper, we illustrate the essential and distinctive features of the OCBC approach, using a simple, yet very sophisticated example, and contrast OCBC with contemporary, case-centric notations. We then relate the approach to recent developments in the conceptual understanding of processes, events, and their constituents, introducing a series of challenges and points of reflections for the community that could be effectively attacked by relying on recent works by Nicola and colleagues.

2. Case-Centric Notations and Many-to-Many Processes

As pointed out in the introduction, contemporary process modelling notations such as BPMN, EPC, and UML activity diagrams typically require a one-to-one correspondence between a *process*[1] and a corresponding notion of *case*. This notion of case represents the main object type targeted (and manipulated) by the process. For this reason, in the remainder of the paper we use the umbrella term *case-centric* to refer to such notations. Examples of cases are orders in order-to-cash e-commerce scenarios, or prospective students in the context of admission processes to university study programs.

The intuition behind this correspondence is that each instance of the process will target (and evolve) a single object of the corresponding case class. Since contemporary notations mainly focus on the process control-flow, that is, on the activities/tasks to be executed and on their acceptable execution orderings, being case-centric also implies that each task execution in a given process instance will target, again, a single case object. A limited form of one-to-many relationship between cases and tasks is supported through the usage of loops and multi-instance constructs in the process. Such constructs usually implicitly indicate the presence of object classes acting as *subcases*, where many subcases may refer to the same case. An example here consists of the various order lines composing an order. The usage of loops or multi-instance constructs in the process is only possible if the evolution of all subcases can be synchronously bound to that of the corresponding case and of the other “sibling" subcases referring to the same case object. This is not obvious, nor feasible in general.

All in all, case-centric notations fall short when managing complex one-to-many relationships, and many-to-many relationships relating different objects that are co-evolved by the same process. Such situations are widespread in real organisations. For example, e-commerce companies like Amazon© flexibly handle order-to-delivery processes by relating multiple customer orders with multiple packages, so that a package sent to a customer may contain a mix of order lines belonging to different orders placed by that customer. This makes it impossible to fix a single notion of case when capturing the process as a whole, since it intrinsically relates objects of different types (such as orders and packages) in a

[1]Recall that, in the BPM literature, *process* is a synonym for *process schema*.

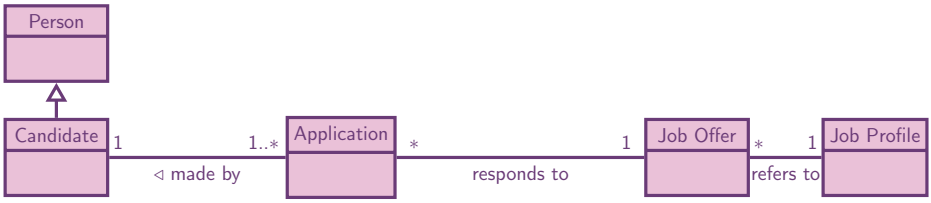


Figure 1. UML class diagram capturing the main object and relationship types of the job hiring domain.

many-to-many fashion. Depending on the subjective view and scope of interest within the process, different case objects may be selected to understand how the process works. For example, a courier may see the order-to-delivery process as centred around the notion of package. A customer, instead, may prefer a view centred on each single order in isolation, ranging from the order placement and payment to its delivery (which is materially realised by delivering all packages containing order lines for that order).

Similarly, in a job hiring process there is a many-to-many relationship between candidates seeking a job, and job offers placed to fill available positions. This requires to decouple the flow of activities under the responsibility of the company placing the job offer, from that followed by each applying candidate. In the following, we take inspiration from [12] and discuss in more details a fragment of a typical job hiring process, pointing out the challenges it pose to case-centric notations.

2.1. A Job Hiring Process

We consider the fragment of a job hiring process enacted by an organisation whenever there is the need of filling an internal position. For simplicity, we consider only two types of actors involved in the process:

- The organisation itself, responsible for the publishing and management of job offers, as well as for the selection of winning applications.
- Candidates interested in the offered positions, who participate to the selection process by registering their personal data and by submitting their applications.

The complexity of the process resides in the fact that it relates many candidates to many job offers, using the key notion of *application* as relator. In the following, we assume that the main object and relationship types of the job hiring domain are those illustrated in the UML class diagram of Figure 1. We illustrate various constraints to describe tasks in the job hiring process together with their mutual temporal relations. For the sake of readability, we use the following fonts and colors:

- boxed, violet sans-serif font to indicate object types;
- violet, underlined sans-serif font to indicate relationship types;

- `blue, bold typewriter font within a rounded rectangle` to indicate tasks;
- *blue, italics font* to highlight temporal aspects, such as the amount of times a task can be repeated, or whether some task is expected to occur before or after another task;
- *green, dash-underlined italics sans-serif font* to point out relationships between tasks and object types.
- *blue, dash dot-underlined italics font* to denote a co-reference indicating which instances of tasks are related via the objects they manipulate.

The relevant constraints for our job hiring example are (see also Figure 7):

C.1 The `register data` task *is about* a Person.

C.2 A Job Offer is *created* by executing the `post offer` task.

C.3 A Job Offer is *closed* by `determining` the `winner`.

C.4 A Job Offer is *stopped* by `canceling` the `hiring`.

C.5 An Application is *created* by executing the `submit` task.

C.6 An Application is *promoted* by `marking` it `as eligible`.

C.7 An Application can be `submitted` only if, *beforehand*, the `data` *about* the Candidate who made *that* Application has been `registered`.

C.8 A `winner` can be `determined` for a Job Offer only if *at least one* Application, responding to *that* Job Offer, has been *previously* `marked as eligible`.

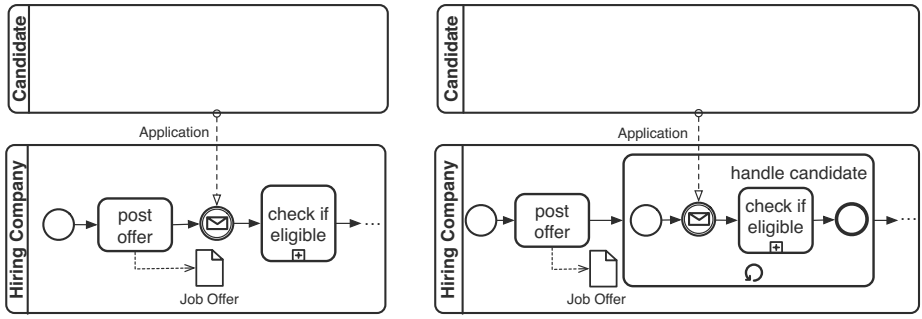
C.9 For each Application responding to a Job Offer, if the Application is `marked as eligible` then a `winner` must be *finally* `determined` for *that* Job Offer, and this last task is executed *only once* for *that* Job Offer.

C.10 When a `winner` is `determined` for a Job Offer, Applications responding to *that* Job Offer *cannot* be `marked as eligible` *anymore*.

C.11 A Job Offer *closed by* a `determine winner` task *cannot* be *stopped* by executing the `cancel hiring` task (and vice-versa).

2.2. Capturing the Job Hiring Example with Case-Centric Notations

The most fundamental issue when trying to capture the job hiring example of Section 2.1 using case-centric notation is to identify *what is the case*. This, in turn, determines what is the orchestration point for the process, that is, *which participant* coordinates process instances corresponding to different case objects.



(a) A job hiring process receiving at most one application

(b) A job hiring process receiving multiple applications in a sequential way; a new application is only handled when the previous applications has been checked for eligibility

Figure 2. Common beginner mistakes when capturing a job hiring process (diagrams inspired from [12])

This problem is apparent when looking at BPMN, which specifies that each process should correspond to a single locus of control, i.e., confined within a single *pool*.[2]

In our example, we have two *participants*: candidates (in turn responsible for managing `Applications`), and the job hiring organisation (in turn responsible for the management of `JobOffers`). However, we cannot use neither of the two to act as unique locus of control for the process: on the one hand, candidates may simultaneously create and manage different applications for different job offers; on the other hand, the organisation may simultaneously spawn and manage different job offers, each one resulting in multiple applications being evaluated. A typical modelling mistake done by novices is to select the job hiring organisation as unique locus of control, and squeeze all tasks therein. This leads to very cumbersome courses of execution as shown in Figure 2.

As clearly pointed out in textbooks (see, e.g., Chapter 8 of [12]), the only way to handle in BPMN a many-to-many process such as the job hiring process considered here, is to distribute the process across multiple, separate pools – in our case study, the candidate *and* the hiring company. Each of such pools focuses on a different case class – in our case study, the hiring company focuses on job offers, whereas the candidate focuses on his/her own applications. However, such multiple pools cannot execute their internal flows in separation, but must instead be properly interconnected using suitable *synchronisation mechanisms*, so as to ensure that the evolution of certain process instance within a pool is properly *aligned* with the evolution of process instances within another pool. For example, in our case study a job offer can be canceled only if no candidate has created an application for it, which also implicitly indicates that once a job hiring has been canceled, none of its applications can be marked as eligible for it. This requires to relate job offers with applications, which can only be done by introducing complex event or data-based synchronisation mechanisms [12] that are not at all mentioned in the description of the process provided in Section 2.1.

[2]Recall that a BPMN pool represents a participant [13].

3. The OCBC Model

We now present the syntax and graphical appearance of OCBC models, and informally comment about their formal semantics. We use the job hiring example of Section 2.1 to illustrate the main concepts and to show the sophistication of this approach. The original proposal of the OCBC model [14, 11] is the way activities and data are related. In particular, an OCBC model captures, at once:

- data dependencies, represented using standard data models containing classes, relationships and constraints between them;
- activities, accounting for units of work within a process;
- *mutual relationships between activities and classes*, linking the execution of activities in a given process with the data objects they manipulate;
- *temporal constraints* between activities;
- *co-reference constraints* that enforce the application of temporal dependencies, and in particular scope their application to those activities instances that indirectly co-refer thanks to the objects and relationships they point to.

We start by recalling data models and temporal constraints, which are then used as the basic building blocks of the OCBC approach.

3.1. The Data Model – ClaM

We assume that data used by the activities is structured according to the ClaM data model (which stands for *CLAss data Model*). While we do not advocate here for a new data model, for simplicity, we assume ClaM to be a simplified version of UML, with 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. More formally we have:

Definition 1 (ClaM Syntax). *A conceptual schema* Σ *in the Class Model, ClaM, is a tuple*

$$\Sigma = (\mathcal{U}_C, \mathcal{U}_R, \tau, \#_{dom}, \#_{ran}, \mathsf{ISA}, \mathsf{ISA_R}, \text{DISJ}, \text{COV}), \quad \textit{where:}$$

- $\mathcal{U}_C$ *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 *of the domain of a relationship.* $\#_{dom}(R, O)$ *is defined only if there is* O_1 *s.t.* $\tau(R) = (O, O_1)$;
- $\#_{ran} : \mathcal{U}_R \times \mathcal{U}_C \nrightarrow \mathbb{N} \times (\mathbb{N} \cup \{\infty\})$ *is a* partial *function defining* cardinality constraints *of the range of a relationship.* $\#_{ran}(R, O)$ *is defined only if there is* O_1 *s.t.* $\tau(R) = (O_1, O)$;

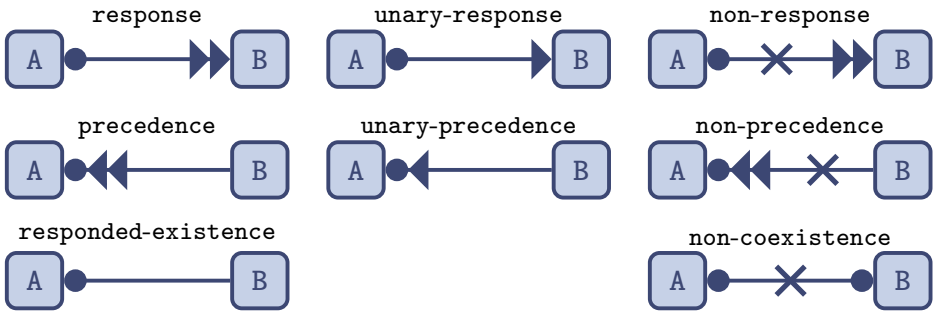


Figure 3. Types of temporal constraints between activities

- $\mathsf{ISA} \subseteq \mathcal{U}_C \times \mathcal{U}_C$ *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*;*
- $\mathsf{ISA}_\mathsf{R} \subseteq \mathcal{U}_R \times \mathcal{U}_R$*, similar to* ISA*, allows to specify* sub-relationships *in the model;*
- $\textsc{disj} \subseteq 2^{\mathcal{U}_C} \times \mathcal{U}_C$ *is a binary relation defining the set of disjoint sub-classes in an* ISA *hierarchy on object classes;*
- $\textsc{cov} \subseteq 2^{\mathcal{U}_C} \times \mathcal{U}_C$ *is a binary relation defining the set of sub-classes covering the super-class in an* ISA *hierarchy on object classes.* □

As for the formal set-theoretic semantics of ClaM and its translation to DLs we refer to [15, 16, 17]. There, 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, while DISJ and COV are interpreted in the obvious way using disjointness/union between the denotation of the involved classes. More specifically, since we are using data models in a dynamic setting where object and relationships evolve over time, we interpret them along the temporal semantics presented in [18, 19, 20] for temporal data models. Essentially, the constraints captured therein must be satisfied in each snapshot (i.e., time point) of the system.

Example 1. Figure 1 can be represented as a ClaM conceptual schema as:

$$\begin{aligned}
\mathcal{U}_C &= \{\mathsf{Person}, \mathsf{Candidate}, \mathsf{Application}, \mathsf{Job\ Offer}, \mathsf{Job\ Profile}\};\\
\mathcal{U}_R &= \{\mathsf{made\ by}, \mathsf{responds\ to}, \mathsf{refers\ to}\};\\
\tau(\mathsf{made\ by}) &= (\mathsf{Application}, \mathsf{Candidate}); \ldots\\
\mathsf{ISA} &= \{(\mathsf{Candidate}, \mathsf{Person})\};\\
\#_{dom}(\mathsf{made\ by}, \mathsf{Application}) &= (1, \infty); \ldots\\
\#_{ran}(\mathsf{made\ by}, \mathsf{Candidate}) &= (1, 1); \ldots
\end{aligned}$$

Note that cardinalities are shown in the diagram of Figure 1 using the UML reading. ◁

`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 execute, 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.

Figure 4. Intuitive meaning of temporal constraints

3.2. Temporal Constraints over Activities

Taking inspiration from the DECLARE patterns [3], we present here the temporal constraints between (pairs of) activities that can be expressed in OCBC. Figure 3 graphically renders such constraints, while their textual representation is defined next.

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`$\}$*, as shown in Figure 3, 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* $A_1, A_2 \in \mathcal{U}_A$. □

In the literature, such constraints are typically formalised using linear temporal logic over finite traces [21, 22]. We report their intuitive meaning in Figure 4.

We observe that the *non-precedence* constraint is syntactic sugar, as it can be emulated using *non-response*:

$$\texttt{non-precedence}(A, B) \equiv \texttt{non-response}(B, A).$$

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`.

3.3. OCBC Models and their Components

We are now ready to define the OCBC model, and comment on its constitutive components, starting from data models and temporal constraints as respectively defined in Sections 3.1 and 3.2.

Definition 3 (OCBC syntax). *An OCBC model, $\mathcal{M}$, is a tuple:*

$$(ClaM, \mathcal{U}_A, \mathcal{U}_{R_{AC}}, \tau_{R_{AC}}, \#_{act}, \#_{obj}, cref, neg\text{-}cref), \text{ where:}$$

- *ClaM is a data model as in Definition 1;*
- $\mathcal{U}_A$ *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* $O \in \mathcal{U}_C$*;*
- $\#_{act} : \mathcal{U}_{R_{AC}} \times \mathcal{U}_A \twoheadrightarrow \mathbb{N} \times (\mathbb{N} \cup \{\infty\})$ *is a* partial function *defining* cardinality constraints *on the participation of activities in activity-object relationships.* $\#_{act}(R, A)$ *is defined only if there is* O *s.t.* $\tau_{R_{AC}}(R) = (A, O)$*;*
- $\#_{obj} : \mathcal{U}_{R_{AC}} \times \mathcal{U}_C \twoheadrightarrow \{1\}$ *is a* partial function *that, when defined, denotes the activity that* generated *a given object in* O*.* $\#_{obj}(R, O)$ *is defined only if there is* O *s.t.* $\tau_{R_{AC}}(R) = (A, O)$*;*
- *cref is the partial function of* co-reference constraints *s.t.*
$$cref : \Sigma^{+}_{TC} \times \mathcal{U}_{R_{AC}} \times \mathcal{U}_{R_{AC}} \twoheadrightarrow \mathcal{U}_C \cup \mathcal{U}_R;$$
- *neg-cref is the partial function of* negative co-reference constraints *s.t.*
$$neg\text{-}cref : \Sigma^{-}_{TC} \times \mathcal{U}_{R_{AC}} \times \mathcal{U}_{R_{AC}} \twoheadrightarrow \mathcal{U}_C \cup \mathcal{U}_R.$$

□

In the following we detail the semantics of an OCBC model by concentrating on the two main aspects, i.e., *activity-object relationships* and *co-reference constraints*.

Activity-object relationships capture how activities relate to classes. Let $R \in \mathcal{U}_{R_{AC}}$ so that $\tau_{R_{AC}}(R) = (A, O)$. The intuitive meaning of R is that each instance of activity A operates over a objects of type O (typically, a single one). In this light, inverses of activity-object relationships are assumed to be functional. On top of this, we single out activity-object relationships capturing the fact that objects of the related class are *generated* when instances of the related activity are executed. We call these special relationships as *generating activity-object relationships* while we denote the task as *object generating task*. If R is o a generating activity-object relationship, then it is associated to a cardinality constraint of the form $\#_{obj}(R, O) = 1$. The semantics of this cardinality constraint is as follows: whenever an object o is of type O at a given time t, then there must have been a previous time t' at which an activity instance a of type A was executed on o (i.e., such that $R(a, o)$ held at time t').

Cardinality constraints for participation of activities in activity-object relationships ($\#_{act}$) are instead captured as classical cardinalities in data models (see [15, 18, 20]) with the intended meaning that a task can manipulate many objects.

In the following, we show how activity-object relationships can be used to capture some of the constraints of our job hiring case study, and comment on interesting properties of the resulting OCBC model.

Example 2. Figure 5 shows how constraints **C.1**–**C.6** from the job hiring case study from Section 2.1 can be captured in OCBC. We maintain the shape, color and font

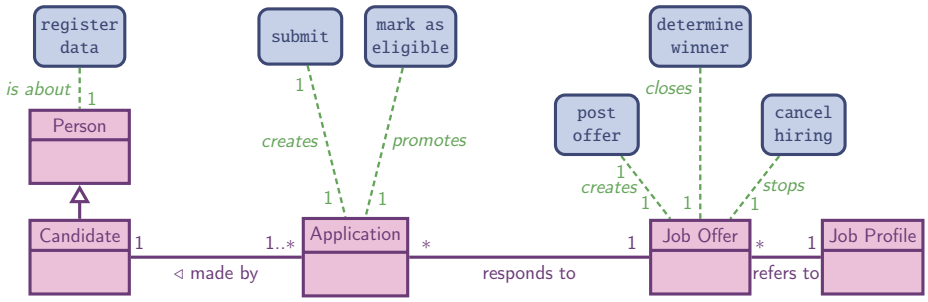


Figure 5. Activity-object relationships in the job hiring scenario of Section 2.1

coding schemes used in Section 2.1, so as to facilitate establishing a connection between the textual description of constraints, and their corresponding OCBC representation.

The activities `submit` and `post offer` are *object generating activities*, and the relationships connecting `submit` to Application and `post offer` to Job Offer are *generating activity-object relationships*. In particular, while a person can exist in the domain even if no data have been registered for him/her, an application can exist in the domain only if it was created by an instance of the submit task (similarly for a job offer).

Notably, even though the OCBC model in Figure 5 does not contain any explicit temporal constraint, the presence of activity-object relationships that generate objects, and their interplay with the constraints present in the data model, already imply the existence of implicit constraints over the allowed activity executions. First and foremost, activities pointing to a class that is also pointed by a generating activity-object relationship, can only be executed on an object if *that very same* object was previously created. This means that an application can be marked as eligible only if it was previously created through the execution of a submit activity instance. Similarly, a job offer can be stopped by a cancel hiring activity instance, or closed by a determine winner activity instance, only if it has been previously created by executing a post offer activity instance.

These temporal dependencies could also propagate further, depending on how the pointed classes are related to each other. We discuss in particular two examples from Figure 5. When a job offer is created by posting it, given the cardinality constraints, it must also refer to exactly one job profile. Thus, such profile must belong to the class Job Profile at the same time when the relationship holds. Even more interesting is the creation of an application, which requires, on the one hand, the candidate owning that application and, on the other hand, a job offer. Thus, due to the interplay between the two generating activity-object relationships for Application and Job Offer, mediated by the responds to relationship linking each application to exactly one job offer (but not viceversa), a complex precedence constraint is implicitly introduced stating that: whenever an Application is `submitted` responding to some Job Offer, *that* Job Offer must have been `posted` *simultaneously* or *before*. ◁

Example 2 already gives an intuition about the sophistication, but also the subtleties, arising when adopting the OCBC approach. This is an intrinsic characteristic of declarative process modelling notations, as extensively discussed in [21, 23].

Co-reference constraints constitute the most powerful construct in OCBC. They have the ability of scoping the temporal constraints defined in Section 3.2, in such a way that they do not apply to *any* instances of the corresponding activities, but only to *those* activity instances that *co-refer* to each other via the objects they refer to. The co-reference may insist on a class, thus requiring such manipulated objects to be the same object, or on a relationship, thus requiring such manipulated objects to be related to each other via that relationship.

This is of utmost importance. Take for example constraint **C.7** from Section 2.1. That constraint is not satisfied if submitting an application is preceded by an arbitrary instance of the `register data` task. The constraint requires that there exists a preceding instance of the `register data` task that *is about* the very same candidate who is related to *that* Application via the made by relationship. Similarly, according to **C.11**, the execution of an instance of the `cancel hiring` task on some Job Offer does not prevent the possibility of `determining` the `winner` for other offers.

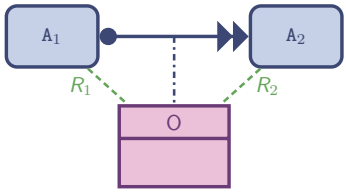
According to Definition 3, there are two kinds of co-reference constraints: positive and negative, and they can range either over object classes (as illustrated in Fig. 6a and 6c) or over relationships (as illustrated in Fig. 6b and 6d). The semantics of co-reference temporal constraints, considering `response` and `not-response` as prototypical example, is informally given in Figure 6. A similar meaning can be assigned to `precedence` and `not-precedence` (by substituting *aftewards* with *before*), and to `responded-existence` and `not-coexistence` (by removing *afterwards* and considering the entire timeline).

To better clarify the usage of such constraints, and also their interesting (and subtle) features, we make again use of our case study in the job hiring domain.

Example 3. The OCBC model illustrated in Figure 7 captures *all* constraints described in Section 2.1. Interestingly, each constraint mentioned in Section 2.1 is mirrored into a corresponding activity-object relationship or co-referenced temporal constraint in the diagram.

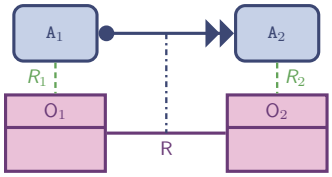
The *co-reference constraints* involving object classes specify constraints on how objects connected to different activities can/cannot be shared. For example, the Job Offer instance *stopped by* a `cancel hiring` activity cannot be the same one *closed by* a `determine winner` activity due to the `non-coexistence` constraint. This means that when one of the two activities is executed on a job offer, the other cannot be executed on the same offer, but may be still executable for other offers. This constraint can be expressed using the following OCBC syntax:

$$\textit{neg-cref}\big(\texttt{non-response}(\texttt{determine winner}, \texttt{cancel hiring}), \textit{closes}, \textit{stops}\big) = \text{Job Offer}$$



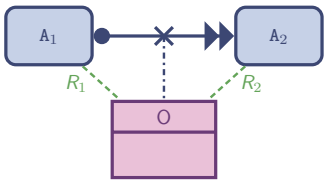
Every time an instance a_1 of $\mathtt{A}_1$ is executed
on some object o of type O (i.e., with $R_1(a_1, o, t_1)$),
then an instance a_2 of $\mathtt{A}_2$ must be executed *afterwards*
on the *same* object o (i.e., with $R_2(a_2, o, t_2)$ and $t_1 < t_2$)

(a) Co-reference of **response** over an object class



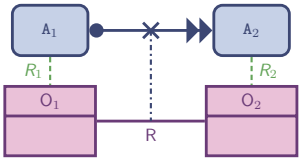
Every time an instance a_1 of $\mathtt{A}_1$ is executed
on some object o_1 of type O_1 (i.e., with $R_1(a_1, o_1, t_1)$),
then an instance a_2 of $\mathtt{A}_2$ must be executed *afterwards*
on some object o_2 of type O_2 (i.e., with $R_2(a_2, o_2, t_2)$ and $t_1 < t_2$)
that relates to o_1 via R at the moment of execution of a_2
(i.e., having $\mathsf{R}(o_1, o_2, t_2)$).

(b) Co-reference of **response** over a relationship



Every time an instance a_1 of A_1 is executed
on some object o of type O (i.e., with $R_1(a_1, o, t_1)$),
then no instance a_2 of $\mathtt{A}_2$ that relates to the *same* object o
can be executed *afterwards* (i.e., with $R_2(a_2, o, t_2)$ and $t_1 < t_2$)

(c) Co-reference of **non-response** over an object class



Every time an instance a_1 of $\mathtt{A}_1$ is executed
on some object o_1 of type O (i.e., with $R_1(a_1, o_1, t_1)$),
then no instance a_2 of $\mathtt{A}_2$ *that* relates to o_1 via R can be executed *afterwards*
(i.e., there is no a_2, o_2 with $R_2(a_2, o_2, t_2)$, $\mathsf{R}(o_1, o_2)$ and $t_1 < t_2$)

(d) Co-reference of **non-response** over a relationship

Figure 6. Co-reference response constraints over (a) object classes and (b) relationships, with their negated versions (c-d)

The co-reference constraints involving relationships specify constraints on how activities can/cannot occur on given objects connected to each other through the data model. As an example, the co-reference **precedence** temporal constraint relating submit to register data enables the possibility of submitting only applications made by candidates who already registered their data. The corresponding OCBC syntax is:

$$\mathit{cref}(\mathtt{precedence}(\mathtt{submit}, \mathtt{register\ data}), \mathit{creates}, \mathit{is\ about}) = \mathsf{made\ by}.$$

Obviously, although not directly linked to Candidate, the *is about* activity-object relationship is inherited by that class given the ISA linking it to Person. It is interesting to notice that this constraint does by no means affect when and how many times the register data task can be executed for a given Person.

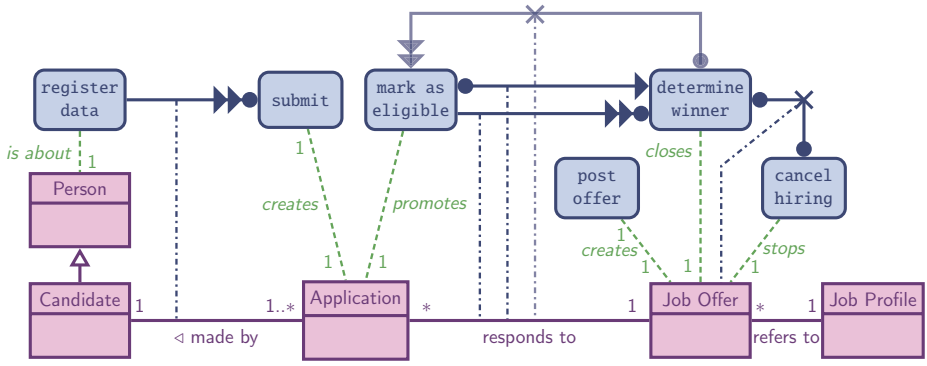


Figure 7. OCBC model for the job hiring scenario of Section 2.1, where each one of **C.1**–**C.11** therein corresponds to either an activity-object relationship or a co-reference temporal constraint in the OCBC model. The lightweight non-response constraint is redundant: it is implied by the other constraints in the diagram.

Indeed, the model implicitly captures the fact that the same Person may update his/her personal data multiple times.

Of particular interest is the `unary-response` constraint that relates `mark as eligible` to `determine winner` (which captures **C.9** from Section 2.1), in particular when *multiple* applications submitted to the same job offer (say, job offer j_o) are marked as eligible. In this case, the constraint requires that every instance of the `mark as eligible` task *promoting* an Application that responds to j_o, is eventually followed by a *single* instance of the `determine winner` task that *closes* offer j_o. Such a single instance of `determine winner` for offer j_o will be actually the *same* for all such eligible applications. In fact, having two distinct instances of `determine winner` for offer j_o would violate the fact that the constraint is a *unary* response. This has a twofold implication:

- The (unique) instance of `determine winner` for j_o must occur *after* all the occurrences of `mark as eligible` for applications that respond to j_o.
- Once the (unique) instance of `determine winner` for j_o is executed, it is no more possible to `mark as eligible` any application responding to j_o. For otherwise, they would require a *later* occurrence of `determine winner` for j_o, which would clash with the uniqueness requirement. This shows that the `non-response` constraint relating `determine winner` to `mark as eligible` via the responds to relationship is actually redundant: it is implied by the `unary-response` constraint relating `mark as eligible` to `determine winner` via the same responds to relationship. Notice that further applications may be still submitted for a closed job offer, but they will not be marked as eligible.
- For similar reasons, the following is also implied: once an eligible Application responds to a Job Offer, then *that* Job Offer *cannot anymore* be

stopped. Indeed, an eligible Application requires that the (unique) Job Offer is eventually *closed* (and thus *cannot anymore* be *stopped* due to the `non-coexistence` constraint between `determine winner` and `cancel hiring`). ◁

4. Conclusion and Discussions

We have presented the OCBC model, which enriches structural conceptual models with behavioural constraints. By means of a small, but relevant case study, we have shown how OCBC captures the interconnection between the process and the data perspective, in a way that allows one to elegantly capture complex many-to-many processes simultaneously operating over different objects. We have discussed that such processes are recurrent in practice, but they can hardly be represented using conventional, case-centric process modelling notations.

We believe that the OCBC approach can constitute the basis for novel research at the intersection of business process management, conceptual modelling, automated reasoning, and formal ontology. A line of research focussed on the discovery of OCBC models from event logs of process executions has been already started [24, 25] . We discuss next some of the main challenges that are still open, and were we believe the synergy with recent works by Nicola and colleagues is essential to effectively attack them.

4.1. Formal semantics and relationship with other process modelling approaches

As already recalled in the introduction, the formal semantics of OCBC can be defined in pure logical terms by resorting to first-order temporal logic, FOL(<), i.e., first-order logic equipped with a special sort that represents time points and the usual $<$ binary relation over time points. In particular, every construct present in the OCBC model of interest is translated into a corresponding FOL(<) formula, and the semantics of the entire model is then simply obtained as the conjunction of all so-obtained formulae.

It is interesting to notice that while this approach is customary in structural conceptual modelling, it departs from the usual approach followed in business process management, where the formal (execution) semantics of the process is given in terms of a Petri net or transition system, whose states can possibly be annotated with the configuration of the data characterising the corresponding state of affairs. This shows that, typically, processes semantically decouples the structural and temporal/dynamic aspects of the system. In addition, if such data have to be interpreted under the assumption of incomplete information, then this semantics cannot capture the evolution of "what is true", but only the evolution of "what is certainly known to be true" [10, 9].

A preliminary attempt to define the formal semantics of OCBC can be found in [11]. One of the main assumptions in that paper is that activity occurrences executed within the process should be identified always by the same object, acting

as a sort of global process (or scenario) identifier. However, this is only needed if one wants to incorporate in the OCBC model also *global temporal constraints* that do not correlate activities via the data model, but just mutually relate all their possible occurrences. If we only consider temporal constraints that co-refer activities via the structural conceptual model, like those presented in this paper, then we do not need such a global identifier, and we can in fact safely assume that each activity occurrence is not important per sè, but only in terms of how it relates to the objects in the data model. This reflects the intuition that a candidate in our job hiring example is interested in knowing the identifier of the application (s)he submitted, but never explicitly refers to the different identifiers of the activity occurrences used to evolve that application.

From the ontological point of view, it remains open to understand whether it is an essential property of activities to refer to the structural conceptual model, and of temporal constraints to co-refer via the structural conceptual model, or whether we should instead conceive also processes that have "global" activities and constraints. This requires a research agenda focused on the nature of processes and their participants, which has been recently initiated by Adamo et al. in [13], but that still misses an ontological characterisation of how data objects participate to processes.

4.2. Reasoning over OCBC Models

It is notoriously known that domain experts have difficulties in understanding declarative process models, both when indicating whether an execution trace conforms to the modelled constraints [23], and when it comes to determine which implicit constraints *implied* by those modelled explicitly [21]. In fact, several hidden dependencies may arise from the (complex) interplay between temporal constraints. At the same time, some modelled constraints may turn out to be redundant, in the sense that they are implied by the other constraints in the model. We have discussed this particular aspect in Example 3. Reasoning becomes therefore essential in the context of OCBC, to support humans in ascertaining properties of the model, as well as in checking whether a given execution trace conforms to the model.

Reasoning over OCBC models presents at least two interesting challenges. First of all, provably correct techniques to carry out fundamental reasoning tasks such as consistency and constraint implication are needed. This requires, first, to understand the boundaries of decidability and complexity for such reasoning tasks. While the semantics in [13] does not directly help towards this goal, we believe that capturing OCBC constraints that always co-refer through the structural conceptual model can lead to a much simpler formalisation. In particular, we are currently working on a formalisation that can be fully captured within the temporal description logics $T_{\mathcal{US}}\mathcal{ALCQI}$ and fragments thereof [26, 27, 28, 29, 30]. Such a formalisation would then provide a concrete basis for automated reasoning, given that the main reasoning tasks for $T_{\mathcal{US}}\mathcal{ALCQI}$ are not only decidable, but also present the same worst-case complexity of reasoning on (non-temporal) $\mathcal{ALCQI}$ knowledge bases.

A second, key aspect is related to a deeper understanding of OCBC in terms of ontological analysis. This requires, as a starting point:

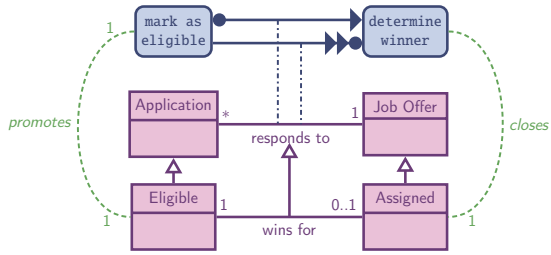


Figure 8. Combining behavioural constraints and subtyping to capture phases and how objects enter in a given phase. The figure modifies the OCBC model for the job hiring example shown in Figure 7, by introducing the explicit characterisation of an Assigned Job Offer. This is a phase entered by a Job Offer when the winner is determined, which in turn requires to pick one Eligible Application among those that respond to that offer. The two temporal constraints now express that when at least one Application that responds to Job Offer *o* is promoted to Eligible, then determine winner closing *o* can (and is expected) to occur exactly once. Since determine winner references the subtype Assigned Job Offer, upon an occurrence of this activity *o* migrates from being just a generic Job Offer, to now be a member of the Assigned phase.

- To study how metaprofiles, in the style of well-known approaches such as ONTOCLEAN [31], apply to OCBC activities, but also how OCBC constraints interact with metaprofiles used in the structural conceptual schema. Figure 8 shows an example of how an OCBC constraint may interact with *phase types*.
- To study how OCBC activities can be conceived as (ontological) *events*. This is a rather unexplored, but extremely important, topic, not just in the context of OCBC, but also within business process management in general. This appears to be extremely difficult when considering foundational ontologies such as DOLCE [32], which adopt an *eternalist* view according to which *all* event occurrences are fully determined (although some of them may be unknown). Fortunately, recent developments by Guizzardi et al. [33] and by Guarino [34] are now putting forward an approach where only past events are considered frozen in time, whereas future events may be genuinely subject to change. This constitutes a solid basis for understanding events in the context of business processes in general, and OCBC in particular.

4.3. Enactment, monitoring, and ontological characterisation of process traces

Understanding the relationship between OCBC models and the general notion of event becomes particularly important when the OCBC model is instantiated into an actual execution. This relationship can be established at a twofold level: the level of activity occurrences, and that of the entire execution trace.

At the activity level, during enactment one has to clearly distinguish the events that have already occurred (and that are consequently frozen in time) from those that may occur in the future. For the latter, it would be interesting

to understand how the foundational approach in [34] can be used to reconstruct the *expectations* about the occurrence of future events. In particular, depending on the current execution trace and the temporal constraints in the model, one should distinguish between activity occurrences that are expected to occur in the future, those that are forbidden to occur in the future, and those that instead may happen. Monitoring goes even beyond that, suggesting that not only future event occurrences have to be considered in terms of these executability criteria, but also that the status of temporal constraints themselves has to be considered [35, 36, 37].

At the overall execution trace level, it has to be noted that there is a fundamental difference between case-centric approaches and declarative, constraint-based approaches such as OCBC. On the one hand, the execution of an instance of a case-based process is, in the terminology of Guarino [34], an *episode*, because it requires to eventually reach one of the ending states foreseen by the process model (e.g., the delivery of an order, or the assignment/cancelation of a job offer). On the other hand, the execution of an instance of a constraint-based process is what Guarino calls in [34] a *process*, since it has no culminating part characterising its full realisation, but can instead be stopped whenever all temporal constraints present in the model are satisfied. In addition, it may happen that an evolving process instance deviates from the acceptable courses of executions foreseen by a case-centric process model, or causes the violation of one or more constraints in a declarative process model (either because of a direct violation of a single constraint, or because of multiple conflicting constraints that could not anymore be all satisfied, no matter how the execution continues [38]). How to characterise these different types of events and episodes is extremely important, also towards analysing processes and their execution in relationship with the broader challenge of organisational compliance, risk, and governance.

References

[1] W. M. P. van der Aalst. *Process Mining: Data Science in activity*. Springer, 2016.

[2] W. M. P. van der Aalst, G. Li, and M. Montali. Object-Centric Behavioral Constraints. *CoRR*, abs/1703.05740, 2017.

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

[4] K. Bhattacharya, C. Gerede, R. Hull, R. Liu, and J. Su. Towards Formal Analysis of Artifact-Centric Business Process Models. In *Proc. of the 11th Int. Enterprise Distributed Object Computing Conf. (EDOC)*, volume 4714 of *LNCS*, pages 288–304. Springer, 2007.

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

[6] P. Gonzalez, A. Griesmayer, and A. Lomuscio. Verification of GSM-Based Artifact-Centric Systems by Predicate Abstractivity. In *Proc. of the 13th Int. Conf. on Service-Oriented Computing (ICSOC)*, volume 9435 of *LNCS*, pages 253–268. Springer, 2015.

[7] V. Vianu. Automatic verification of database-driven systems: a new frontier. In *Proc. of the 12th Int. Conf. on Database Theory (ICDT)*, pages 1–13, 2009.

[8] D. Calvanese, G. De Giacomo, and M. Montali. Foundations of data-aware process analysis: A database theory perspective. In *Proc. of 32nd Symp. on Principles of Database Systems (PODS)*. ACM Press, 2013.

[9] B. Bagheri Hariri, D. Calvanese, M. Montali, G. De Giacomo, R. De Masellis, and P. Felli. Description logic Knowledge and Action Bases. *Journal of Artificial Intelligence Research*, 46, 2013.

[10] H. J. Levesque. Foundations of a functional approach to knowledge representation. *Artificial Intelligence*, 1984.

[11] W. M. P. van der Aalst, A. Alessandro, M. Montali, and S. Tritini. Object-centric behavioral constraints: Integrating data and declarative process modelling. In Alessandro Artale, Birte Glimm, and Roman Kontchakov, editors, *Proc. of the 30th Int. Ws. on Description Logics DL-17*, volume 1879 of *CEUR*. CEUR-WS.org, 2017.

[12] B. Silver. *BPMN Method and Style*. Cody-Cassidy, 2nd edition, 2011.

[13] G. Adamo, S. Borgo, C. Di Francescomarino, C. Ghidini, N. Guarino, and E.M. Sanfilippo. Business processes and their participants: An ontological perspective. In Floriana Esposito, Roberto Basili, Stefano Ferilli, and Francesca A. Lisi, editors, *Proc. of the XVIth Int. Conf. of the Italian Association for Artificial Intelligence (AI*IA)*, volume 10640 of *LNCS*, pages 215–228. Springer, 2017.

[14] W. M. P. van der Aalst, G. Li, and M. Montali. Object-centric behavioral constraints. Corr technical report, arXiv.org e-Print archive, 2017.

[15] D. Berardi, D. Calvanese, and G. De Giacomo. Reasoning on UML class diagrams. *Artificial Intelligence*, 168(1–2):70–118, 2005.

[16] A. Artale, D. Calvanese, R. Kontchakov, V. Ryzhikov, and M. Zakharyaschev. Reasoning over extended ER models. In *Proc. of the 26th Int. Conf. on Conceptual Modeling (ER)*, volume 4801 of *LNCS*, pages 277–292. Springer, 2007.

[17] E. Franconi, A. Mosca, and D. Solomakhin. ORM2: formalisation and encoding in OWL2. In *Proc. of Int. Workshop on Fact-Oriented Modeling (ORM)*, pages 368–378, 2012.

[18] A. Artale, C. Parent, and S. Spaccapietra. Evolving objects in temporal information systems. *Annals of Mathematics and Artificial Intelligence*, 50(1–2):5–38, 2007.

[19] A. Artale and E. Franconi. Foundations of temporal conceptual data models. In *Conceptual Modeling: Foundations and Applications*, volume 5600 of *LNCS*, pages 10–35. Springer, 2009.

[20] A. Artale, R. Kontchakov, V. Ryzhikov, and M. Zakharyaschev. Complexity of reasoning over temporal data models. In *Proc. of the 29th Int. Conf. on Conceptual Modeling (ER)*, volume 4801 of *LNCS*, pages 277–292. Springer, 2010.

[21] M. Montali, M. Pesic, W. M. P. van der Aalst, F. Chesani, P. Mello, and S. Storari. Declarative specification and verification of service choreographiess. *ACM Trans. on the Web (TWEB)*, 4(1), 2010.

[22] G. De Giacomo, R. De Masellis, and M. Montali. Reasoning on LTL on finite traces: Insensitivity to infiniteness. In *Proc. of the 28th AAAI Conference on Artificial Intelligence*, pages 1027–1033, 2014.

[23] D. Fahland, D. Lübke, J. Mendling, H. A. Reijers, W. Barbara, M. Weidlich, and S. Zugal. Declarative versus imperative process modeling languages: The issue of understandability. In Terry A. Halpin, John Krogstie, Selmin Nurcan, Erik Proper, Rainer Schmidt, Pnina Soffer, and Roland Ukor, editors, *Proc. of the 10th Int. Ws. on Business-Process and Information Systems Modeling (BPMDS)*, volume 29 of *LNBIP*, pages 353–366. Springer, 2009.

[24] G. Li, R. Medeiros de Carvalho, and W. M. P. van der Aalst. Automatic discovery of object-centric behavioral constraint models. In *Proc. of the 20th Int. Conf. on Business Information Systems (BIS 2017)*, volume 288 of *LNBIP*, pages 43–58. Springer, 2017.

[25] G. Li, E. G. L. de Murillas, R. Medeiros de Carvalho, and W. M. P. van der Aalst. Extracting object-centric event logs to support process mining on databases. In *Proc. of Information Systems in the Big Data Era (CAiSE Forum)*, volume 317 of *LNBIP*, pages 182–199. Springer, 2018.

[26] F. Wolter and M. Zakharyaschev. Temporalizing description logics. In *Frontiers of Combining Systems*, pages 379 – 401. Research Studies Press-Wiley, 2000.

[27] A. Artale, E. Franconi, F. Wolter, and M. Zakharyaschev. A temporal description logic for reasoning over conceptual schemas and queries. In *Proc. of the 8th European Conference on Logics in Artificial Intelligence (JELIA-02)*, volume 2424 of *LNAI*, pages 98–110. Springer-Verlag, September 2002.

[28] D. Gabbay, A. Kurucz, F. Wolter, and M. Zakharyaschev. *Many-dimensional modal logics: theory and applications*. Studies in Logic. Elsevier, 2003.

[29] C. Lutz, F. Wolter, and M. Zakharyaschev. Temporal description logics: A survey. In *Proc. of the 15th Int. Symposium on Temporal Representation and Reasoning (TIME 08)*, pages 3–14. IEEE Computer Society, 2008.

[30] A. Artale, R. Kontchakov, V. Ryzhikov, and M. Zakharyaschev. A cookbook for temporal conceptual data modelling with description logics. *ACM Transactivity on Computational Logic (TOCL)*, 15(3), 2014.

[31] N. Guarino and C. Welty. Evaluating ontological decisions with ontoclean. *Commun. ACM*, 45(2):61–65, 2002.

[32] A. Gangemi, N. Guarino, C. Masolo, A. Oltramari, and L. Schneider. Sweetening ontologies with DOLCE. In Asunción Gómez-Pérez and V. Richard Benjamins, editors, *Proceedings of the 13th Int. Conf. on Knowledge Engineering and Knowledge Management (EKAW)*, volume 2473 of *LNCS*, pages 166–181. Springer, 2002.

[33] G. Adamo, S. Borgo, C. Di Francescomarino, C. Ghidini, N. Guarino, and E.M. Sanfilippo. Business process activity relationships: Is there anything beyond arrows? In Mathias Weske, Marco Montali, Ingo Weber, and Jan vom Brocke, editors, *Proceedings of the 2018 Business Process Management Forum (BPM Forum)*, volume 329 of *LNBIP*, pages 53–70. Springer, 2018.

[34] N. Guarino. On the semantics of ongoing and future occurrence identifiers. In Heinrich C. Mayr, Giancarlo Guizzardi, Hui Ma, and Oscar Pastor, editors, *Proceedings of the 36th Int. Conf. on Conceptual Modeling (ER)*, volume 10650 of *LNCS*, pages 477–490. Springer, 2017.

[35] A. Bauer, M. Leucker, and C. Schallhart. Comparing LTL Semantics for Runtime Verification. *Logic and Computation*, 2010.

[36] G. De Giacomo, R. De Masellis, M. Grasso, F. M. Maggi, and M. Montali. Monitoring business metaconstraints based on LTL & LDL for finite traces. In Shazia Wasim Sadiq, Pnina Soffer, and Hagen Völzer, editors, *Proceedings of the 12th International Conference on Business Process Management (BPM)*, volume 8659 of *LNCS*, pages 1–17. Springer, 2014.

[37] R. De Masellis, F. M. Maggi, and M. Montali. Monitoring data-aware business constraints with finite state automata. In He Zhang, LiGuo Huang, and Ita Richardson, editors, *Proceedings of the International Conference on Software and System Process (ICSSP)*, pages 134–143. ACM Press, 2014.

[38] F. M. Maggi, M. Westergaard, M. Montali, and W. M. P. van der Aalst. Runtime verification of LTL-based declarative process models. In Sarfraz Khurshid and Koushik Sen, editors, *Proceedings of the 2nd International Conference on Runtime Verification (RV)*, volume 7186 of *LNCS*, pages 131–146. Springer, 2012.