Object-centric Processes with Structured Data and Universal Synchronization

Formal Modelling and Conformance Checking

Alessandro Gianola[1], Marco Montali[2], and Sarah Winkler[2]

[1] INESC-ID/IST, Universidade de Lisboa, Portugal
alessandro.gianola@tecnico.ulisboa.pt
[2] Free University of Bozen-Bolzano, Italy
{montali,winkler}@inf.unibz.it

Abstract. Real-world processes often involve interdependent objects, also carrying on sophisticated data values, such as integers, reals, strings. However, existing process formalisms fall short to combine key modeling features, such as tracking object identities, supporting complex datatypes, handling dependencies among them, and object-aware synchronization. Object-centric Petri nets with identifiers (OPIDs) partially address these needs but treat objects as unstructured identifiers (e.g., order and item IDs), overlooking the rich semantics of complex data values (e.g., item prices or other attributes). To overcome these limitations, we introduce Data-aware OPIDs (DOPIDs), a framework that strictly extends OPIDs by incorporating structured data manipulation capabilities, and full synchronization mechanisms. In spite of the expressiveness of the model, we show that it can be made operational, considering in particular the task of conformance checking. Specifically, we define a novel operational approach leveraging satisfiability modulo theories (SMT) to compute data-aware object-centric alignments.

Keywords: Object-centric conformance checking · Universal Synchronization · Data-aware Processes · Complex Datatypes · SMT

1 Introduction

In recent years, research in information systems supporting the execution of business and work processes has increasingly emphasized the need for multi-perspective process models. Prominently, the goal has been to tackle the intricate relationship between the control flow and the data with which this control flow can interact. This calls for investigating how data influence the process behavior, and how the control flow of the process impacts on the data queried and manipulated by activities and decision points. Data dimensions can be manifold: case variables carrying data, multi-case interactions with objects involved in many-to-many relations, and interactions with persistent storage like a relational database. In this spectrum, a growing stream of research is producing different formal models of data-aware processes with the twofold aim of supporting

representation and *computation*, on the one hand covering relevant modelling constructs, on the other providing effective algorithmic techniques for process analysis and mining (such as, most prominently, automated discovery and conformance checking). Within this stream, two distinct directions emerged. The first focuses on enriching case-centric processes by incorporating structured case attributes (e.g., the price of a product, the age and name of a customer, as well as more complex structures such as a persistent relational storage). This, in turn, supports expressing how activities in the process read and write these variables, and how decision points use these variables to express routing conditions for cases. A prime example in this vein is that of Data Petri nets (DPNs [17,15]). The second direction has instead the purpose of lifting the case-centricity assumption, tackling so-called object-centric processes where multiple objects, interconnected via complex one-to-many and many-to-many relationships, are co-evolved by the process (e.g., orders containing multiple products, shipped in packages that may mix up products from different orders). It has been pointed out that straightjacketing this complexity through a single case notion yields to misleading process analysis and mining results [1,4]. Several process modelling formalisms have emerged, such as object-centric Petri nets [3], synchronous proclets [8], and different variants of Petri nets with identifiers (PNIDs) [19,11,22,13], tackling essential features like: *(i)* tracking objects, by representing concurrent flows of object ids, and by enabling parallel progression of object ids such as package shipments and order notifications; *(ii)* object creation and manipulation, handling dependencies among objects and their related data values, and supporting one-to-one and one-to-many relationships - e.g., adding multiple items to an order or splitting it into separate packages. *(iii)* object-aware full synchronization, i.e., an object can flow through an activity only if some (subset) or all (exact) related objects also flow – ensuring that an object can proceed through an activity only when certain conditions are met (e.g., initiating order billing only when some or all associated packages have been delivered).

The ultimate goal of this work is to reconcile these two lines of research, proposing a unified formalism supporting at once the most advanced modelling features for object-centric processes, as well as those dealing with data attributes and conditions. Specifically, we provide a twofold contribution in *representation* and *computation*. As for representation, we start from OPIDs [13] – the most sophisticated formalism based on PNIDs: they support all the main object-centric required modelling features, with the exception of universal/exact synchronization. We lift OPIDs into a new class of PNIDs called DOPIDs, which at once close the gap regarding synchronization, and support infusing in the net also data values with a variety of data types, together with conditions expressed over such data values, ranging from simple comparison conditions to advanced forms of aggregation. As for computation, we consider conformance checking, showing that we can lift existing SMT-techniques for conformance checking to cover all the newly introduced features. We do so by integrating and extending the SMT-encodings for conformance checking separately studied for OPIDs [13] and DPNs [9], covering the full DOPID spectrum.

	object creation	object removal	concurrent object flows	multi-object transfer	multi-object spawning	object relations	subset sync	exact sync	coreference	struct. data	object reference	conformance
OC nets [3]	✓	✓	✓	✓	✓	✗	✗	✗	✗	✗	imp.	[16]
synchronous proclets [8]	✓	✓	✓	∼	✓	✓	✓	✓	✗	✗	imp.	✗
DPNs [15]	✓	✓	✓	✗	✗	✗	✗	✗	✗	✓	imp.	[17]
PNIDs [19,11,22]	✓	✓	✓	∼	∼	✓	∼	✗	✓	✗	exp.	✗
OPIDs [13]	✓	✓	✓	✓	✓	✓	✓	∼	✓	✗	exp.	[13]
DABs [6]	✓	✓	✗	✗	✗	✗	✗	✗	✗	✓	no	✗
DOPIDs	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	exp.	*here*

Table 1. Comparison of Petri net-based object-centric process modelling languages along main modelling features, tracking which approaches support conformance. ✓ indicates full, direct support, ✗ no support, and ∼ indirect support.

2 Related Work and Modelling Features

To highlight key modeling features of DOPIDs, we reviewed relevant literature on case-centric data-aware processes [10,6,12] and object-centricity [1,4,2]. A summary of these features and their implementation across different approaches is presented in Tab. 1.

The first key feature is the incorporation of constructs for *creating and deleting objects*. Different approaches vary based on whether objects are *explicitly referenced* within the model or are only *implicitly manipulated*. Another critical aspect is the ability for objects to *flow concurrently* and independently — for example, items being picked while their corresponding order is paid (see *divergence* in [1]). Additionally, models may support the *simultaneous transfer of multiple objects* of the same type, such as processing several items in a single transaction. Moreover, transitions in these models must account for the manipulation of multiple objects, either of the same type or different types, at the same time (see *convergence* in [1]). A type of convergence occurs when a single transition *generates an unbounded number of child objects* from a parent object, where the children are all linked to the parent. For instance, placing an order can create unboundedly many associated items. Once this parent-child *one-to-many relationship* is established, other forms of convergence, such as synchronizing transitions, can be introduced. These transitions allow a parent object to evolve only if *some* (*subset synchronization*) or *all* its child objects (*exact synchronization*) are in a certain state. In addition, advanced *coreference* techniques can be employed to simultaneously examine and evolve multiple interconnected objects. Finally, an essential feature is the support for advanced and *structured data types*, such as integers, reals, lists, and arrays. They enhance the objects manipulated by the process with additional information, allowing users to define complex conditions and constraints that act as guards for the transitions in the process model, possibly enriching it with background knowledge. This

final feature, unlike the others, is more characteristic of data-aware extensions of *case-centric* process models [18], where the focus is traditionally placed on the complex structure of data values, often governed by relational logical theories [7]. Among these approaches, the most advanced framework is the DAB model [6], which supports rich forms of database-driven data, and sophisticated forms of reasoning. In process mining, multi-perspective models capable of incorporating richer data representations while expressing concurrent flows have also been introduced. A prominent example is Data Petri Nets [10], a Petri net-based formalism that, while more expressive, remains case-centric.

Regarding object-centric models, several Petri net-based formal models have been introduced [21,3,8,19,11,22]. Object-centric nets [3] offer an implicit approach to object-centricity. Here, places and transitions are associated with different object types. Simple arcs match with a single object at a time, while double arcs handle arbitrarily many objects of the same type. However, the lack of mechanisms to track object relationships prevents modeling object synchronization and coreference. Alignment-based conformance checking for this approach is in [16]. Synchronous proclets [8] offer a framework that can implicitly express the tracking of objects and their mutual relationships. They include specialized constructs to support the described types of convergence, including subset and exact synchronization, though other forms of coreference are not supported. Multi-object transfers are approximated through iteration, processing objects one by one. Conformance checking for proclets is implicitly tackled here for the first time (considering that DOPIDs generalize proclets). Petri nets with identifiers (PNIDs), and variants, have been studied in [19,11,22], though without addressing conformance checking. PNIDs build upon ν-Petri nets by explicitly managing objects and their relationships through identifier tuples. Unlike object-centric nets and proclets, PNIDs lack constructs for manipulating unboundedly many objects in a single transition. As demonstrated in [11], multi-object transfers, spawning, and subset synchronization can be achieved through object coreference and iterative operations. Nevertheless, exact synchronization would necessitate data-aware wholeplace operations, which no variant of PNID supports. Finally, DOPIDs strictly subsume OPIDS [13], extending them with complex, structured data capabilities and by supporting full object-aware synchronization. In fact, OPIDs do not allow guards in transitions, and only permit unstructured object ids and subset synchronization.

3 Object-Centric Event Logs with Data Attributes

We start from object-centric event logs as in [3,13], and enrich them with data attributes. To this end, we assume that data types are divided in two classes: a class Σ_{obj} of *object id* types, and a class Σ_{val} of *data-value* types.

Consistently with the literature, every object id type $\sigma \in \Sigma_{obj}$ has an (uninterpreted) domain $dom(\sigma) \subseteq \mathcal{O}$, given by all object ids in $\mathcal{O}$ of type σ. Such identifiers are used to refer to objects in the real world, and can be compared only for equality and inequality. Examples are order and product identifiers.

To capture the data attributes attached to objects and recorded in event logs, such as for example the price of a product and the delivery address of an order, we also introduce *data-value domains* for data-value types in Σ_{val}: $dom(\texttt{bool}) = \mathbb{B}$, the booleans; $dom(\texttt{int}) = \mathbb{Z}$, the integers; $dom(\texttt{rat}) = \mathbb{Q}$, the rational numbers; and $dom(\texttt{string}) = \mathbb{S}$, the strings over some fixed alphabet; unconstrained finite sets $dom(\texttt{finset}_i) = \mathbb{D}_i$, for some finite set $\mathbb{D}_i$ (subsuming strings); all the defined types are equipped with relational and function symbols defined over it. While boolean, numerical, and string values are of standard, relational and function symbols are particularly important not only towards generality of the approach, but also to conceptually capture implicit boolean, numerical, and string values that are attached to objects but not explicitly manipulated by the process. For example, for an order management process where every order gets attached to a delivery address (explicitly accounted and manipulated by the process) and an owner (implicitly assumed, but not directly manipulated by the process), one may opt for modelling the delivery address as a string, and the owner as a function taking an order id as its only argument.

In addition to the sets Σ_{obj} and Σ_{val} for object and data-value types, fix a set $\mathcal{A}$ of activities and a set $\mathbb{T}$ of timestamps with a total order $<$. We also consider (partial) assignments from a set of variables $\mathcal{V}$ to elements of their domain. The set of all such assignments is denoted $Assign^{\mathcal{V}}$.

Definition 1. *An* event log (with objects and attributes) *is a tuple* $L = \langle E, \mathcal{O}, \pi_{act}, \pi_{obj}, \pi_{time}, \pi_{val} \rangle$ *where: (i)* E *is a set of event identifiers, (ii)* $\mathcal{O}$ *is a set of object identifiers that are typed by a function* $\texttt{type}\colon \mathcal{O} \to \Sigma_{obj}$*, (iii) the functions* $\pi_{act}\colon E \to \mathcal{A}$*,* $\pi_{obj}\colon E \to \mathcal{P}(\mathcal{O})$*, and* $\pi_{time}\colon E \to \mathbb{T}$ *associate each event* $e \in E$ *with an activity, a set of affected objects, and a timestamp, respectively, such that for every* $o \in \mathcal{O}$ *the timestamps* $\pi_{time}(e)$ *of all events* e *such that* $o \in \pi_{obj}(e)$ *are all different.(iv) the function* $\pi_{val}\colon E \to Assign^{\mathcal{V}}$*. This function associates each event* $e \in E$ *with a set of affected data-values.*

Given an event log and an object $o \in \mathcal{O}$, we write $\pi_{trace}(o)$ for the tuple of events involving o, ordered by timestamps. Formally, $\pi_{trace}(o) = \langle e_1, \ldots, e_n \rangle$ such that $\{e_1, \ldots, e_n\}$ is the set of events in E with $o \in \pi_{obj}(e)$, and $\pi_{time}(e_1) < \cdots < \pi_{time}(e_n)$ (by assumption these timestamps can be totally ordered.) In examples, we often leave $\mathcal{O}$, $\mathcal{A}$ and domains implicit and present an event log L as a set of tuples $\langle e, \pi_{act}(e), \pi_{obj}(e), \pi_{time}(e), \pi_{val}(e) \rangle$ representing events. Timestamps are shown as natural numbers, and concrete event ids as $\#_0, \#_1, \ldots$.

Example 1. We consider a more sophisticated variant of the order-to-shipment example in [13], in turn inspired from [3]. Instead of emphasizing the problems related to reshuffling products from one order to the other, which were tackled in [13] and are homogeneously handled in this work, we focus on distinctive features of our extended model, not covered in object-centric nets [3] nor OPIDs [13]. Process executions in this scenario involve the following events: *(i)* place order creates an order indicating its id, the number of days expected by the customer for the delivery, and the order content, consisting of one or more products; *(ii)* pay cc and pay bt pay the order by credit card and bank transfer respectively;

(iii) pick item fetches a product from the warehouse; *(iv)* ship takes an order with its products and deals with the shipment, also declaring the shipment mode.

Consider $\mathcal{O} = \{o_1, p_1, p_2\}$ with $\mathtt{type}(o_1) = \mathit{order}$ and $\mathtt{type}(p_1) = \mathtt{type}(p_2) = \mathit{product}$. The following is an event log with four events $E = \{\#_0, \#_1, \#_2, \#_3\}$, indicating that order o_1 is placed with two products p_1, p_2 and with 3 days for preferred delivery, then o_1 is paid, p_1 is picked, and finally o_1 is shipped only with p_1, confirming the 3 days and selecting truck as mode:

$$\langle \#_0, \mathsf{place\ order}, \{o_1, p_1, p_2\}, \{d \mapsto 3\}, 1\rangle, \langle \#_1, \mathsf{pay\ cc}, \{o_1\}, \emptyset, 4\rangle,$$
$$\langle \#_2, \mathsf{pick\ item}, \{o_1, p_1\}, \emptyset, 5\rangle, \langle \#_3, \mathsf{ship}, \{o_1, p_1\}, \{d \mapsto 3, s \mapsto \mathsf{truck}\}, 9\rangle$$

The notions of object and trace graphs from [3,13] remain identical also in our setting. In particular, the notion of trace graph, used to single out all the events and flows referring to an object, does not change as what matters for correlating events is that they insist on the same objects, while it is not important whether they share data values. In the example, we have a single trace graph for o_1.

4 Data-Aware Object-Centric Petri Nets with Identifiers

We define *data-aware object-centric Petri nets with identifiers* (DOPIDs), enriching the features of OPIDs [13] with data values, complex data manipulation capabilities, and full object-aware synchronization.

As in PNIDs and OPIDs, objects can be created in DOPIDs using ν variables, and tokens can carry object ids, data values, or tuples containing multiple object ids and/or data values. The latter account at once for relationships among objects, and attributes connecting objects to data values. So, objects can be linked to other objects or data values, e.g., as in Ex. 3, where a product tracks the order it belongs to, and an order is associated to its shipment mode. Arcs are labeled with (tuples of) variables to match with objects and relations.

In the style of object-centric nets [3] and synchronous proclets [8], in a DOPID one can spawn and transfer multiple objects at once, using an extension of the mechanism introduced for OPIDs, based on the usage of special "set variables" that match with *sets of objects*. The refinement consists in the possibility of indicating, when operating over a set of objects by consuming multiple tokens at once, whether one wants to consume *some* or *all* matching objects. The latter case could not be tackled in OPIDs, and is essential to reconstruct the different forms of synchronization present in synchronous proclets.

A second, key extension to OPIDs is that DOPIDs support different types of data values carried by tokens, i.e., *object ids* to account for the identifiers of objects and *structured data-values* such as integers or reals. By expressing guards over such data values, DOPIDs incorporate and go beyond the modelling features of Data Petri nets [17,15] and related formalisms.

Formal Definition. Let $\Sigma = \Sigma_{obj} \uplus \Sigma_{val}$ be the overall set of types (including both object and data value types). A set of *variables* $\mathcal{V}$ is *typed* if there is a function $\mathtt{type}\colon \mathcal{V} \to \Sigma$ assigning a type to each variable in $\mathcal{V}$. In addition to the types in Σ, we also consider list types with a base type $\sigma \in \Sigma$, denoted as $[\sigma]$.

As in colored Petri nets, each place has a *color*: a cartesian product of data types from Σ. More precisely, the set of colors Col is the set of all $\sigma_1 \times \cdots \times \sigma_m$ such that $m \geq 1$ and $\sigma_i \in \Sigma$ for all $1 \leq i \leq m$. We fix a set of Σ-typed variables $\mathcal{X} = \mathcal{V} \uplus \mathcal{V}_{list} \uplus \mathcal{V}^{\subseteq}_{list} \uplus \mathcal{V}^{=}_{list} \uplus \Upsilon$ as the disjoint union of:

1. a set $\mathcal{V}$ of "normal" variables that refer to single objects or data values, denoted by lower-case letters like v, with a type $\texttt{type}(v) \in \Sigma$;
2. a set $\mathcal{V}_{list}$ of list variables referring to a list of objects of the same type, denoted by upper case letters like U, with $\texttt{type}(U) = [\sigma]$ for some $\sigma \in \Sigma$;
3. two sets $\mathcal{V}^{\subseteq}_{list}$ and $\mathcal{V}^{=}_{list}$ that contain *annotated* list variables $U^{\subseteq}$ and $U^{=}$ resp., for each list variable U in $\mathcal{V}_{list}$ – these will be used to express whether *some* or *all* objects matching the variable must be considered;
4. a set Υ of variables ν referring to fresh objects, with $\texttt{type}(\nu) \in \Sigma_{obj}$.

We assume that infinitely many variables of each kind exist, and for every $\nu \in \Upsilon$, that $dom(\texttt{type}(\nu))$ is infinite, for unbounded supply of fresh objects [20]).

In DOPIDs, tokens are tuples of object ids and data-values, each associated with a color. E.g., for the objects in Ex. 1, we want to use $\langle o_1 \rangle$, $\langle o_1, p_1 \rangle$, and $\langle o_1, 5 \rangle$ as tuples carried by tokens – respectively representing a reference to: order o_1, the relationship indicating that product p_1 is contained in order o_1, and the fact that o_1 has 5 days of desired delivery by the customer. To define relationships between objects in consumed and produced tokens when firing a transition, we need, as customary, *inscriptions* of arcs, which are tuples of variables in $\mathcal{X}$.

Definition 2. *An* inscription *is* $\boldsymbol{v} = \langle v_1, \ldots, v_m \rangle$ *s.t.* $m \geq 1$ *and* $v_i \in \mathcal{X}$ *for all* i*, but at most one* $v_i \in \mathcal{V}_{list} \uplus \mathcal{V}^{\subseteq}_{list} \uplus \mathcal{V}^{=}_{list}$ *(*$1 \leq i \leq m$*). We call* $\boldsymbol{v}$ *a* transfer-template inscription *if* $v_i \in \mathcal{V}_{list}$*,* $\subseteq$-template inscription *if* $v_i \in \mathcal{V}^{\subseteq}_{list}$*, or* $=$-template inscription *if* $v_i \in \mathcal{V}^{=}_{list}$ *for some* i*, and a* simple inscription *otherwise.*

For instance, for $o, p \in \mathcal{V}$ and $P \in \mathcal{V}_{list}$, there are inscriptions $\langle o, P \rangle$ or $\langle p \rangle$, the former being a template inscription and the latter a simple one. Inscription $\langle o, P^{=} \rangle$ is a $=$-template inscription since it contains the variable $P^{=}$ in $\mathcal{V}^{=}_{list}$. However, $\langle P, P \rangle$ is not a valid inscription as it has two list variables. By allowing at most one list variable in inscriptions, we restrict to many-to-one relationships between objects. Recall that many-to-many relationships can be modeled as many-to-one with auxiliary objects, through reification.

As we will see later, simple, $\subseteq$- and $=$-template inscriptions will be used when consuming tokens, while simple and transfer-template inscriptions will be employed when producing tokens. Specifically, template inscriptions will be used to capture an arbitrary number of tokens of the same color: intuitively, if o is of type *order* and P of type [*product*], then $\langle o, P \rangle$ refers to a single order with an arbitrary number of produced product ids (i.e., other object ids). When consuming tokens carrying order-product pairs from a place, $\langle o, P^{\subseteq} \rangle$ selects *some* tokens carrying different products for a given order o, while $\langle o, P^{=} \rangle$ selects *all* of them. This will be useful to tackle subset and exact synchronization (cf. Section 2).

We define the color of an inscription $\iota = \langle v_1, \ldots, v_m \rangle$ as the tuple of the types of the involved variables, i.e., $\texttt{color}(\iota) = \langle \sigma_1, \ldots, \sigma_m \rangle$ where $\sigma_i = \texttt{type}(v_i)$ if $v_i \in \mathcal{V} \cup \Upsilon$, and $\sigma_i = \sigma'$ if v_i is a list variable of type $[\sigma']$. Moreover, we set

$vars(\iota) = \{v_1, \ldots, v_m\}$. E.g. for $\iota = \langle o, P\rangle$ with o, P as above, we have $\texttt{color}(\iota) = \langle order, product\rangle$ and $vars(\iota) = \{o, P\}$. The same applies to $\subseteq$- and $=$-template inscriptions. The set of all inscriptions is denoted Ω.

To define guards on transitions, we consider the following definition of *conditions*, where we assume that (uninterpreted) functions and relations are defined over Σ (i.e., all object id and data value domains):

Definition 3. *Given $\mathcal{V}$ and $\mathcal{V}_{list}$, a* constraint *c and* expressions *s, n, r, d, k, t_D,* and *t_K are defined by the grammar:*

$$c ::= v_b \mid b \mid d = d \mid k \geq k \mid k > k \mid R(d, \ldots, d) \mid R(k, \ldots, k) \mid c \wedge c \mid \neg c$$
$$n ::= v_n \mid z \mid sum(Z) \mid min(Z) \mid max(Z) \mid n + n \mid -n$$
$$r ::= v_r \mid q \mid sum(Q) \mid min(Q) \mid max(Q) \mid mean(Q) \mid r + r \mid -r$$
$$s ::= v_s \mid h \mid f(s, \ldots, s) \qquad d ::= s \mid f_w(d, \ldots, d) \mid f_w(k, \ldots, k)$$
$$k ::= n \mid r \mid g_w(k, \ldots, k) \mid g_w(d, \ldots, d) \mid k + k \mid -k \mid sum(t_K)$$
$$t_D ::= D \mid f_y(t_D) \mid f_y(t_K) \qquad t_K ::= Z \mid Q \mid g_y(t_K) \mid g_y(t_D)$$

where $v_b, v_s, v_n, v_r \in \mathcal{V}$, $\texttt{type}(v_b) = \texttt{bool}$, $b \in \mathbb{B}$, $\texttt{type}(v_n) = \texttt{int}$, $z \in \mathbb{Z}$, $\texttt{type}(v_r) = \texttt{rat}$, $q \in \mathbb{Q}$, $\texttt{type}(v_s) = \texttt{finset}_i$, $h \in \mathbb{D}_i$ (some i), $Z, Q, D \in \mathcal{V}_{list}$, $\texttt{type}(Z) = [\texttt{int}]$, $\texttt{type}(Q) = [\texttt{rat}]$, D has non-arithmetic type, f_w, f_y are functions with arithmetic codomains, g_w, g_y are functions with non-arithmetic ones.

This definition may seem quite involved, but it defines esentially simple concepts. Term s defines strings as variables, constants, or inductive function applications. Term n defines integers, sums of integers, or aggregators *sum*, *min*, and *max* applied to lists of integers. Term r is analogous to n but for rational numbers, for which also the aggregator *mean* is defined. Terms k and d define, with a double induction, mixed terms that can combine different types: the only difference is that the root symbol for k lives in a arithmetical domain ($\mathbb{Z}$ or $\mathbb{Q}$), wheres for d it lives in a non-arithmetical domain. An analogous double induction defines the list terms t_D and t_K, which are built on list variables (a function applied to a list term is applied component-wise, and returns another list), but differ in the fact that t_K lives in a arithmetical domain. Notice also that the term k can be produced by the application of the aggregator *sum* to a list variable t_k. Standard equivalences apply, hence disjunction (i.e., $\vee$) of constraints can be used, as well as comparisons $=, \neq, <, \leq$ on integer and rational expressions.

The constraints defined in Def. 3 serve to express conditions on the values of (list) variables. We stress again the expressiveness of this language, which makes also use of *aggregators*, such as the sum of all elements in a finite list.

Example 2. We consider two constraints that will be later used to express conditions on transitions in Ex. 3. Consider an integer variable d representing the expected maximum days for a delivery by a customer, and a string variable m denoting the shipment mode of an order. Condition $(d \leq 5 \wedge m = \mathsf{car}) \vee (d > 5 \wedge m = \mathsf{truck})$ expresses that either d is at most 5 days and the shipment mode is car, or that d is 6 days or more and the shipment mode is truck.

Consider now a list variable P for products. We use a unary function *cost* that returns the cost of each product in euros, a rational number. This expresses the background knowledge that every product has a cost, but this is not explicitly manipulated at the process level (so it will not appear in the log). Consistently with Def. 3, $cost(P)$ represents the list of rational numbers that contains the costs of all elements in P. Then, constraint $sum(cost(P)) \leq 1000$ expresses that the overall cost of all products in P does not exceed 1000 euros.

Definition 4. *A* data-aware object-centric Petri net with identifiers *(DOPID) is a tuple* $N = (\Sigma_{obj}, \Sigma_{val}, P, T, F_{in}, F_{out}, \texttt{color}, \ell, \texttt{guard})$*, where:*

1. *P and T are finite sets of places and transitions such that $P \cap T = \emptyset$;*
2. *$\texttt{color}\colon P \to Col$ maps every place to a color over Σ_{obj}, and Σ_{val};*
3. *$\ell\colon T \to \mathcal{A} \cup \{\tau\}$ is the transition labelling where τ marks an invisible activity,*
4. *$F_{in}\colon P \times T \to \Omega$ is a partial function called* input flow*, such that $\texttt{color}(F_{in}(p,t)) = \texttt{color}(p)$ for every $(p,t) \in dom(F_{in})$;*
5. *$F_{out}\colon T \times P \to \Omega$ is a partial function called* output flow*, such that $\texttt{color}(F_{out}(t,p)) = \texttt{color}(p)$ for every $(t,p) \in dom(F_{out})$;*
6. *$\texttt{guard} : T \to \{\varphi \mid \varphi \in \mathcal{C}(\mathcal{X})\}$ is a partial guard assignment function, s.t., for every $\texttt{guard}(t) = \varphi$ and $t \in T$, $Vars(\varphi) \subseteq vars_{in}(t) \cup vars_{out}(t) \cup \Upsilon$, where $vars_{in}(t) = \cup_{p \in P} Vars(F_{in}(p,t))$ and $vars_{out}(t) = \cup_{p \in P} Vars(F_{out}(t,p))$*

As a well-formedness condition, we assume that in F_{in} one can only use simple, $\subseteq$-template and $=$-template inscriptions, while in F_{out} one can only use simple and transfer-template inscriptions (cf. Def. 2).

Simple flows (i.e., flows with simple inscriptions) are meant to consume and produce (tuples of) single matching tokens, whereas template flows (i.e., flows with template inscriptions) to consume and produce multiple (tuples of) matching tokens. Consumption in this case can be fine-tuned by indicating whether some (in the case of a $\subseteq$-template inscription) or all (for a $=$-template inscription) matching tokens have to be consumed. Production transfers such matched tokens to the corresponding output places, using transfer-template inscriptions.

As it will be clear from the next example and the definition of the semantics of DOPIDs, this is used to capture variable arcs in [16] as in OPIDs, but also to reconstruct different forms of synchronizations. In particular, $=$-template inscriptions realize a form of data-aware wholeplace operation: they do not consume all the tokens contained in a place, but all those that match the inscription.

For a DOPID N as in Def. 4, we also use the common notations $\bullet t = \{p \mid (p,t) \in dom(F_{in})\}$ and $t\bullet = \{p \mid (t,p) \in dom(F_{out})\}$.

Ex.3 shows some of the most important features of DOPIDs, and how they exceed by far the modelling capabilities of existing object-centric formalisms.

Example 3. Fig. 1 graphically depicts a DOPID for a simple yet sophisticated order-to-ship process, extending the running example from [13]. Variables ν_o of type *order* and ν_p of type *product*, both in Υ, refer to new orders and products. Normal variables $o, p \in \mathcal{V}$ of type *order* and *product* refer to existing orders and products, and variable P of type $[product]$ to lists of products. We also use two data value variables d and m, resp. of type integer and string, to represent the

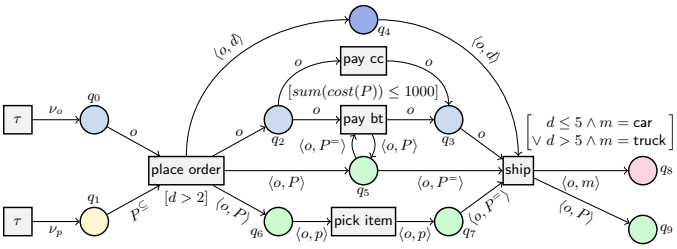


Fig. 1. DOPID of an order-to-ship process.

delivery days desired by the customer, and the shipment mode. For readability, single-component tuples are written without brackets (e.g., o instead of $\langle o \rangle$).

We explain the model transition by transition. The two silent transitions on the left have the purpose of injecting (fresh) orders and products in the net. Transition place order takes an order o from place q_0 and *some* available products P from place q_1, and assign all those products to the order. This is done through the output transfer-template inscription $\langle o, P \rangle$, which transfers to place q_2 $|P|$ tokens, each carrying a pair $\langle o, p \rangle$ with p taken from P. In this respect, place q_5 explicitly represents what in proclets is called correlation set: for every active order in the system, which products belong to it. When place order fires for a given order o, also place q_6 is with products of o, but from there single products will be consumed concurrently and independently through pick item. In addition to assigning products to the consumed order o, place order also assigns to o the maximum number of expected days of delivery d by the customer, inserting the pair $\langle o, d \rangle$ in place q_4, responsible for tracking this attribute for every active order. Since d is only used in output flows, this reconstructs what in DPNs is called a "write" variable. Also notice that this write is constrained by condition $d > 2$, capturing that d can only take values above 2 days. Finally, place order changes the state of the picked order o, moving it place q_0 to q_2

From q_2, two transitions can be fired for o, reflecting two payment modes. Specifically, order o either flows through pay cc, or through pay bt, but the latter can only be selected if the overall cost o (i.e., the sum of the costs of all its products) does not exceed 1000 euros (cf. Ex. 2). To obtain all the products of o, pay bt needs to fetch those products from place q_5, using inscription $\langle o, P^{=} \rangle$. This inscription requires that the first component matches order o consumed from place q_2, while $P^{=}$ forces all the matching pairs for o to be fetched from q_5. As the purpose is to use such products, but not to remove the corresponding pairs, they are all transferred back to q_5 using the inscription $\langle o, P \rangle$.

Concurrently with order payment, the state of single products (recalling their order) is changed when they are, one by one, picked via the pick item transition.

Finally, the ship transition is enabled for a paid order o under the following conditions. First and foremost o can be shipped only if *all* its products have been picked. This is expressed through the two $=$-template input flows, both using the same inscription $\langle o, P^{=} \rangle$. At once, this has the effect of consuming all pairs

containing products of o from places q_5 and q_7. The output transfer-template inscription $\langle o, P\rangle$ connecting ship to place q_9 has then the effect of transferring all those pairs there. At the same time, ship considers the value d for o, and through the attached condition (cf. Ex. 2) determines the shipmpent mode for o, which is recorded in place q_8, linking m to o using inscription $\langle o, m\rangle$.

Two important remarks are in place wrt. Ex. 3. First, the modelling pattern in Fig. 1 that employs the "correlation" place q_5 to keep track of products contained in an order, paired with the two =-template inscriptions in shipment to ensure that *all* products of an order have been actually picked, is what makes DOPIDs able to support universal synchronization in the full generality of synchronous proclets [8], something that was out of reach until now for formal models based on PNIDs. Second, as DOPIDs handle multiple objects and data values at once, they are not only able to express read-write conditions and guards as in DPNs [15], both also to express more sophisticated conditions using aggregation expressions.

Semantics. Given the set of object ids $\mathcal{O}$ and a set of data-values $\mathcal{DV}$, the set of *tokens* $\boldsymbol{\mathcal{TOK}}$ is the set of tuples of object ids and data-values $\boldsymbol{\mathcal{TOK}} = \{(\mathcal{O} \uplus \mathcal{DV})^m \mid m \geq 1\}$. The *color* of a token $\omega \in \boldsymbol{\mathcal{TOK}}$ of the form $\omega = \langle o_1, \ldots, o_j, dv_{j+1}, \ldots, dv_m\rangle$ is denoted $\mathtt{color}(\omega) = \langle \mathtt{type}(o_1), \ldots, \mathtt{type}(o_j), \mathtt{type}(dv_{j+1}), \ldots, \mathtt{type}(dv_m)\rangle$. To define the execution semantics, we first introduce a notion of a *marking* of an DOPID $\mathcal{N} = \langle \Sigma_{obj}, \Sigma_{val}, P, T, F_{in}, F_{out}, \mathtt{color}, \ell, \mathtt{guard}\rangle$, namely as a function $M\colon P \to 2^{\boldsymbol{\mathcal{TOK}}}$, such that for all $p \in P$ and $\langle o_1, \ldots, o_j, dv_{j+1}, \ldots, dv_m\rangle \in M(p)$, it holds that $\mathtt{color}(\langle o_1, \ldots, o_j, dv_{j+1}, \ldots, dv_m\rangle) = \mathtt{color}(p)$. Let $\mathit{Lists}(\mathcal{O})$ denote the set of object lists of the form $[o_1, \ldots, o_k]$ with $o_1, \ldots, o_k \in \mathcal{O}$ such that all o_i have the same object type; the type of such a list is then $[\mathtt{type}(o_1)]$. Analogously, given $\mathit{Lists}(\mathcal{DV})$ the set of data-value lists $[dv_1, \ldots, dv_l]$ with $dv_1, \ldots, dv_l \in \mathcal{DV}$ such that all dv_i have the same data-value type, the type of such a list is $[\mathtt{type}(dv_1)]$. Next, we define *bindings* to fix which data are involved in a transition firing.

Definition 5. *A* binding *for a transition t and a marking M is a type-preserving function $b\colon \mathit{vars}_{in}(t) \cup \mathit{vars}_{out}(t) \to (\mathcal{O} \cup \mathit{Lists}(\mathcal{O})) \uplus (\mathcal{DV} \cup \mathit{Lists}(\mathcal{DV}))$, such that for all $U \in \mathcal{V}_{list}$, we have $b(U) = b(U^{=}) = b(U^{\subseteq})$. To ensure freshness of created values, we demand that b is injective on $\Upsilon \cap \mathit{vars}_{out}(t)$, and that $b(\nu)$ does not occur in M for all $\nu \in \Upsilon \cap \mathit{vars}_{out}(t)$.*

E.g., for transition ship in Ex. 3 the mapping b that sets $b(o) = o_1$ and $b(P) = [p_1, p_2, p_3]$ is a binding. Next, we extend bindings to inscriptions to fix which tokens (not just single objects) participate in a transition firing. The extension of a binding b to inscriptions, i.e., variable tuples, is denoted $\boldsymbol{b}$. For an inscription $\iota = \langle v_1, \ldots, v_m\rangle$, let $o_i = b(v_i)$ for all $1 \leq i \leq m$. Then $\boldsymbol{b}(\iota)$ is the set of object tuples defined as follows: if ι is a simple inscription then $\boldsymbol{b}(\iota) = \{\langle o_1, \ldots, o_m\rangle\}$. Otherwise, there must be one v_i, $1 \leq i \leq n$, such that $v_i \in \mathcal{V}_{list}$, and consequently o_i must be a list, say $o_i = [u_1, \ldots, u_k]$ for some $u_1, \ldots, u_k$. Then $\boldsymbol{b}(\iota) = \{\langle o_1, \ldots, o_{i-1}, u_1, o_{i+1}, \ldots, o_m\rangle, \ldots, \langle o_1, \ldots, o_{i-1}, u_k, o_{i+1}, \ldots, o_m\rangle\}$. The set of all bindings is denoted by $\mathcal{B}$. We define that a transition with a binding b

is enabled if all object tuples pointed by $\boldsymbol{b}$ occur in the current marking. Given a condition ϕ, $\boldsymbol{b}(\phi)$ means that b is applied to all variables occurring in ϕ.

Definition 6. *A transition $t \in T$ and a binding b for marking M are* enabled *in M if (1) $b(guard(t))$ is true; (2) $\boldsymbol{b}(F_{in}(p,t)) \subseteq M(p)$ for all $p \in \bullet t$, such that $F_{in}(p,t)$ is a (plain) variable flow, a $\subseteq$-variable or a non-variable flow; (3) $\boldsymbol{b}(F_{in}(p,t)) = M(p)$ for all $p \in \bullet t$ such that $F_{in}(p,t)$ is a $=$-variable flow.*

E. g., the binding b with $b(o) = o_1$ and $b(P) = [p_1, p_2, p_3]$ is enabled in a marking M of the DOPID in Ex. 3 with $\langle o_1 \rangle \in M(q_{blue})$ and $\langle o_1, p_1 \rangle, \langle o_1, p_2 \rangle, \langle o_1, p_3 \rangle \in M(q_{green})$, for q_{blue} and q_{green} the input places of ship with respective color.

Definition 7. *Let transition t be enabled in marking M with binding b. The* firing *of t yields the new marking M' given by $M'(p) = M(p) \setminus \boldsymbol{b}(F_{in}(p,t))$ for all $p \in \bullet t$, and $M'(p) = M(p) \cup \boldsymbol{b}(F_{out}(p,t))$ for all $p \in t\bullet$.*

We write $M \xrightarrow{t,b} M'$ to denote that t is enabled with binding b in M, and its firing yields M'. A sequence of transitions with bindings $\rho = \langle (t_1, b_1), \ldots, (t_n, b_n) \rangle$ is called a *run* if $M_{i-1} \xrightarrow{t_i, b_i} M_i$ for all $1 \leq i \leq n$, in which case we write $M_0 \xrightarrow{\rho} M_n$. For such a binding sequence ρ, the *visible subsequence* ρ_v is the subsequence of ρ consisting of all (t_i, b_i) such that $\ell(t_i) \neq \tau$.

An *accepting* object-centric Petri net with identifiers is an object-centric Petri net $\mathcal{N}$ together with a set of initial markings M_{init} and a set of final markings M_{final}. For instance, for Ex. 3, M_{init} consists only of the empty marking, whereas M_{final} consists of all (infinitely many) markings in which each of the two rightmost places has at least one token, and all other places have no token. The *language* of the net is given by $\mathcal{L}(\mathcal{N}) = \{\rho_v \mid m \xrightarrow{\rho} m',\ m \in M_{init}, \text{ and } m' \in M_{final}\}$, i.e., the set of visible subsequences of accepted sequences.

The next example relates an observed event log with a DOPID, preluding to the conformance checking problem tackled in the next section.

Example 4. The event log described in Ex. 1 cannot be suitably replayed in the DOPID N of Fig. 1, due to two mismatches: according to N, o_1 must be shipped by car (as the preferred days are below 5), and with both products p_1 and p_2. This in turn requires that, before shipping, also product p_1 must be picked.

5 Alignment-Based Conformance Checking for DOPIDs

We tackle conformance following alignment-based approaches for object-centric processes [16,13], which relate trace graphs to model runs to find deviations. In the remainder of the section, we consider a trace graph T_X and an accepting DOPID N, assuming that the language of N is not empty. We define moves as in [16,13], with the difference that they also contain assignments of data values.

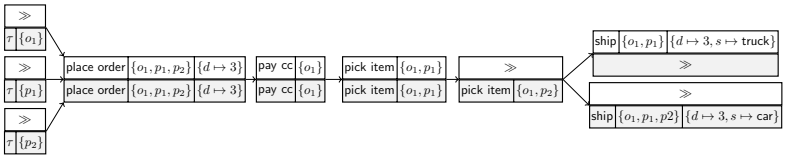
Definition 8 (Moves). *A* model *move is a tuple in $\{\gg\} \times ((\mathcal{A} \cup \{\tau\}) \times \mathcal{P}(\mathcal{O}) \times Assign^{\mathcal{V}})$, a* log move *a tuple in $(\mathcal{A} \times \mathcal{P}(\mathcal{O}) \times Assign^{\mathcal{V}}) \times \{\gg\}$, and a* synchronous *move is of the form $\langle\langle a, O_M, \alpha_M \rangle, \langle a', O_L, \alpha_L \rangle\rangle \in (\mathcal{A} \times \mathcal{P}(\mathcal{O}) \times Assign^{\mathcal{V}})^2$ such that $a = a'$ and $O_L = O_M$. The set of all synchronous, model, and log moves over X and N is denoted moves(T_X, N).*

In the object-centric setting, an alignment is a *graph* of moves G. We use the notions of log projection $G|_{log}$ and model projection $G|_{mod}$ exactly as defined in [16,13]. Intuitively, the log projection is a graph that restricts to the log component of moves, omitting skip symbols. Edges are as in G, except that one adds edges that "shortcut" over model moves, i.e. where the log component is $\gg$; the model projection is analogous for the other component. For formal definitions cf. [16,13]. Next we define an alignment as a graph over moves where the log and model projections are a trace graph and a run, respectively.

Definition 9 (Alignment). *An* alignment *of a trace graph T_X and an accepting DOPID N is an acyclic directed graph $\Gamma = \langle C, B\rangle$ with $C \subseteq moves(T_X, N)$ such that $\Gamma|_{log} = T_X$, there is a run $\rho = \langle\langle t_1, b_1\rangle, \ldots, \langle t_n, b_n\rangle\rangle$ with $\rho_v \in \mathcal{L}(N)$, and the model projection $\Gamma|_{mod} = \langle C_m, B_m\rangle$ admits a bijection $f\colon \{\langle t_1, b_1\rangle, \ldots, \langle t_n, b_n\rangle\} \to C_m$ such that*

- *if $f(t_i, b_i) = \langle a, O_M, \alpha_M\rangle$ then $\ell(t_i) = a$, $O_M = range(b_i) \cap \mathcal{O}$, and $\alpha_M = b_i|_{\mathcal{V}_{val}}$ for all $1 \le i \le n$, where $\mathcal{V}_{val}$ is the subset of $\mathcal{V}$ with a sort in Σ_{val};*
- *for all $\langle r, r'\rangle \in B_m$ there are $1 \le i < j \le n$ such that $f(t_i, b_i) = r$ and $f(t_j, b_j) = r'$, and conversely, if $r, r' \in C_m$ with $r = f(t_i, b_i)$ and $r' = f(t_{i+1}, b_{i+1})$ for some $1 \le i < n$ then $\langle r, r'\rangle \in B_m$.*

Example 5. Below is an alignment Γ for the log in Ex. 1 wrt. N in Ex. 3. The log (resp. model) component is shown on top (resp. bottom) of moves.



We adopt the cost function from [16], but we need to extend it to account for mismatching data values; other definitions are, however, possible as well.

Definition 10. *The cost of a move M is defined as follows: (1) if M is a log move $\langle\langle a_L, O_L, \alpha_L\rangle, \gg\rangle$ then $cost(M) = |O_L|$, (2) if M is a model move $\langle\gg, \langle a_M, O_M, \alpha_M\rangle\rangle$ then $cost(M) = 0$ if $a_{mod} = \tau$, and $cost(M) = |O_M|$ otherwise, (3) if M is a synchronous move $\langle\langle a_L, O_L, \alpha_L\rangle, \langle a_M, O_M, \alpha_M\rangle\rangle$ then $cost(M)$ is the number of variables in $dom(\alpha_L) \cup dom(\alpha_M)$ for which α_L and α_M differ. For an alignment $\Gamma = \langle C, B\rangle$, we set $cost(\Gamma) = \sum_{M \in C} cost(M)$, i.e., the cost of an alignment Γ is the sum of the cost of its moves.*

E.g., Γ in Ex. 5 has cost 7, as it involves one log move (cost 2) and two non-silent model moves (costs 2 and 3). In fact, Γ is optimal:

Definition 11. *An alignment Γ of a trace graph T_X and an accepting DOPID $\mathcal{N}$ is* optimal *if $cost(\Gamma) \le cost(\Gamma')$ for all alignments Γ' of T_X and $\mathcal{N}$.*

The *conformance checking task* for an accepting DOPID $\mathcal{N}$ and a log L is to find optimal alignments with respect to $\mathcal{N}$ for all trace graphs in L.

SMT encoding for conformance checking. We assume a given DOPID N and a trace graph T_X. An SMT encoding of the conformance checking task for DOPIDs can be done in a very similar way as for OPIDs as presented in [13]. For reasons of space, we focus on the differences here.

First, in encoding-based conformance checking, it is essential to fix upfront an upper bound on the size of an optimal alignment [9]. For DOPIDs, we can exploit [13, Lemma 1]: DOPIDs differ from OPIDs in the presence of data and synchronization, but this does not affect the reasoning of that proof. We thus get an upper bound n on the number of nodes in the model projection of the alignment and an upper bound K on the number of objects used in a transition. Using T_X we can get a finite set of objects $O \subseteq \mathcal{O}$ that contains all objects in the alignment. Let moreover m be the number of nodes in T_X. From N we can also read off the maximum number D of data values involved in a transition.

The encoding uses similar SMT variables as in [13], namely (a) transition variables $\mathtt{T}_j$ for all $1 \leq j \leq n$ to encode the j-th transition in the run; (b) marking variables $\mathtt{M}_{j,p,\boldsymbol{o}}$ for every time point $0 \leq j \leq n$, every place $p \in P$, and every vector $\boldsymbol{o}$ of objects with elements in O to encode markings; (c) to encode the actual length of the run, we use an integer variable $\mathtt{len}$; and (d) to keep track of which objects are used by transitions, we use object variables $\mathtt{O}_{j,k}$ for all $1 \leq j \leq n$ and $0 \leq k \leq K$ to encode which objects populate inscriptions, and (e) distance variables $\delta_{i,j}$ to optimize the cost of the alignment, similar as in [9,13]. In addition, to keep track of which data values are used by transitions, we use (e) data value variables $\mathtt{d}_{j,k}$ for all $1 \leq j \leq n$ and $0 \leq k \leq D$ to encode which data values are used in which transition.

There are then two main differences in the encoding in [13]. First, when encoding transitions, conditions need to be taken into account, similarly to [9], using object variables $\mathtt{O}_{j,k}$ and data variables $\mathtt{d}_{j,k}$. Uninterpreted predicates and function symbols are directly modeled as such in SMT, while numeric predicates and aggregation functions are natively supported by SMT solvers. Second, to model synchronization, in contrast to the subset synchronization employed in [13] it must be ensured that inscription variables from $\mathcal{V}^{=}_{list}$ are always instantiated by *all* tokens currently in the respective places.

Notably, we can show that by calling the SMT solver with all accumulated constraints, we obtain an assignment to SMT variables from which the optimal alignment can be decoded. See [14] for details.

6 Conclusions

We have introduced DOPIDs, a new process formalism that unifies modelling features of case-centric data-aware processes and object-centric processes, especially offering an object-centric paradigm with full synchronization and support for complex data. We also showed a novel operational approach leveraging the SMT technology to tackle alignment-based conformance checking for DOPIDs. In future work, we intend to conduct an experimental evaluation of this approach, and study discovery techniques for DOPIDs.

References

1. van der Aalst, W.M.P.: Object-centric process mining: Dealing with divergence and convergence in event data. In: Proc. SEFM (2019)
2. van der Aalst, W.M.P.: Twin transitions powered by event data - using object-centric process mining to make processes digital and sustainable. In: Joint Proc. ATAED/PN4TT (2023)
3. van der Aalst, W.M.P., Berti, A.: Discovering object-centric Petri nets. Fundam. Informaticae **175**(1-4), 1–40 (2020)
4. Berti, A., Montali, M., van der Aalst, W.M.P.: Advancements and challenges in object-centric process mining: A systematic literature review. CoRR **abs/2311.08795** (2023)
5. Boltenhagen, M., Chatain, T., Carmona, J.: Optimized SAT encoding of conformance checking artefacts. Computing **103**(1), 29–50 (2021)
6. Calvanese, D., Ghilardi, S., Gianola, A., Montali, M., Rivkin, A.: Formal modeling and smt-based parameterized verification of data-aware BPMN. In: Proc. of BPM 2019. LNCS, vol. 11675, pp. 157–175. Springer (2019), https://doi.org/10.1007/978-3-030-26619-6_12
7. Damaggio, E., Deutsch, A., Hull, R., Vianu, V.: Automatic verification of data-centric business processes. In: Proc. of BPM 2011. Lecture Notes in Computer Science, vol. 6896, pp. 3–16. Springer (2011). https://doi.org/10.1007/978-3-642-23059-2_3
8. Fahland, D.: Describing behavior of processes with many-to-many interactions. In: Proc. PETRI NETS (2019)
9. Felli, P., Gianola, A., Montali, M., Rivkin, A., Winkler, S.: Data-aware conformance checking with SMT. Inf. Syst. **117**, 102230 (2023)
10. Felli, P., de Leoni, M., Montali, M.: Soundness verification of data-aware process models with variable-to-variable conditions. Fundam. Informaticae **182**(1), 1–29 (2021). https://doi.org/10.3233/FI-2021-2064
11. Ghilardi, S., Gianola, A., Montali, M., Rivkin, A.: Petri net-based object-centric processes with read-only data. Inf. Syst. **107**, 102011 (2022)
12. Gianola, A.: Verification of Data-Aware Processes via Satisfiability Modulo Theories, Lecture Notes in Business Information Processing, vol. 470. Springer (2023). https://doi.org/10.1007/978-3-031-42746-6
13. Gianola, A., Montali, M., Winkler, S.: Object-centric conformance alignments with synchronization. In: Proc. 36th CAiSE. LNCS, vol. 14663, pp. 3–19. Springer (2024). https://doi.org/10.1007/978-3-031-61057-8_1
14. Gianola, A., Montali, M., Winkler, S.: Object-centric processes with structured data and universal synchronization (extended version) (2024), available from https://www.inf.unibz.it/montali/papers/dopid-long-version.pdf
15. de Leoni, M., Felli, P., Montali, M.: A holistic approach for soundness verification of decision-aware process models. In: ER. LNCS, vol. 11157, pp. 219–235. Springer (2018)
16. Liss, L., Adams, J.N., van der Aalst, W.M.P.: Object-centric alignments. In: Proc. ER (2023)
17. Mannhardt, F., de Leoni, M., Reijers, H.A., van der Aalst, W.M.P.: Balanced multi-perspective checking of process conformance. Computing **98**(4), 407–437 (2016)
18. Montali, M., Calvanese, D.: Soundness of data-aware, case-centric processes. Int. J. Softw. Tools Technol. Transf. **18**(5), 535–558 (2016). https://doi.org/10.1007/S10009-016-0417-2

19. Polyvyanyy, A., van der Werf, J.M.E.M., Overbeek, S., Brouwers, R.: Information systems modeling: Language, verification, and tool support. In: Proc. CAiSE (2019)
20. Rosa-Velardo, F., de Frutos-Escrig, D.: Decidability problems in petri nets with names and replication. Fundam. Informaticae **105**(3), 291–317 (2010)
21. Sommers, D., Sidorova, N., van Dongen, B.: Aligning event logs to resource-constrained ν-petri nets. In: Proc. PETRI NETS. LNCS, vol. 13288, pp. 325–345 (2022)
22. van der Werf, J.M.E.M., Rivkin, A., Polyvyanyy, A., Montali, M.: Data and process resonance - identifier soundness for models of information systems. In: Proc. PETRI NETS (2022)

A Encoding

We detail the encoding outlined in the main body of the paper. The encoding crucially relies on the following bound on the number of objects and the number of moves in an optimal alignment, taken from [13]. DOPIDs differ from the OPIDs in [13] by the presence of data and synchronization, but this does not affect the reasoning of this proof.

Lemma 1. *Let $\mathcal{N}$ be a DOPID and $T_X = \langle E_X, D_X \rangle$ a trace graph with optimal alignment Γ. Let $m = \sum_{e \in E_X} |\pi_{obj}(e)|$ the number of object occurrences in E_X, and $c = \sum_{i=1}^{n} |dom(b_i)|$ the number of object occurrences in some run ρ of $\mathcal{N}$, with $\rho_v = \langle \langle t_1, b_1 \rangle, \ldots \langle t_n, b_n \rangle \rangle$. Then $\Gamma|_{mod}$ has at most $(|E_X| + c + m)(k+1)$ moves if $\mathcal{N}$ has no ν-inscriptions, and at most $(|E_X| + 3c + 2m)(k+1)$ otherwise, where k is the longest sequence of silent transitions without ν-inscriptions in $\mathcal{N}$. Moreover, $\Gamma|_{mod}$ has at most $2c+m$ object occurrences in non-silent transitions.*

Next, we detail which variables are necessary for the SMT encoding.

Variables. We start by fixing the set of variables used to represent the (unknown) model run and alignment:

(a) Transition variables $\mathtt{T}_j$ of type integer for all $1 \leq j \leq n$ to identify the j-th transition in the run. To this end, we enumerate the transitions as $T = \{t_1, \ldots, t_L\}$, and add the constraint $\bigwedge_{j=1}^{n} 1 \leq \mathtt{T}_j \leq L$, with the semantics that $\mathtt{T}_j$ is assigned value l iff the j-th transition in ρ is t_l.
(b) To identify the markings in the run, we use marking variables $\mathtt{M}_{j,p,\boldsymbol{o}}$ of type boolean for every time point $0 \leq j \leq n$, every place $p \in P$, and every vector $\boldsymbol{o}$ of objects with elements in O such that $\mathtt{color}(\boldsymbol{o}) = \mathtt{color}(p)$. The semantics is that $\mathtt{M}_{j,p,\boldsymbol{o}}$ is assigned true iff $\boldsymbol{o}$ occurs in p at time j.
(c) To keep track of which objects are used by transitions of the run, we use object variables $\mathtt{O}_{j,k}$ of type integer for all $1 \leq j \leq n$ and $0 \leq k \leq K$ with the constraint $\bigwedge_{j=1}^{n} 1 \leq \mathtt{O}_{j,k} \leq |O|$. The semantics is that if $\mathtt{O}_{j,k}$ is assigned value i then, if $i > 0$ the k-th object involved in the j-th transition is o_i, and if $i = 0$ then the j-th transition uses less than k objects.

In addition, we use the following variables to represent alignment cost:

(d) Distance variables $\delta_{i,j}$ of type integer for every $0 \leq i \leq m$ and $0 \leq j \leq n$, their use will be explained later.

Constraints. We use the following constraints on the variables defined above:

(1) *Initial markings.* We first need to ensure that the first marking in the run ρ is initial. By the expression $[\boldsymbol{o} \in M(p)]$ we abbreviate $\top$ if an object tuple $\boldsymbol{o}$ occurs in the $M(p)$, and $\bot$ otherwise.

$$\bigvee_{M \in M_{init}} \bigwedge_{p \in P} \bigwedge_{\boldsymbol{o} \in \boldsymbol{O}_{\mathtt{color}(p)}} \mathtt{M}_{0,p,\boldsymbol{o}} = [\boldsymbol{o} \in M(p)] \qquad (\varphi_{init})$$

(2) *Final markings.* Next, we state that after at most n steps, but possibly earlier, a final marking is reached.

$$\bigvee_{0 \leq j \leq n} \bigvee_{M \in M_{final}} \bigwedge_{p \in P} \bigwedge_{\boldsymbol{o} \in \boldsymbol{O}_{\mathtt{color}(p)}} \mathtt{M}_{j,p,\boldsymbol{o}} = [\boldsymbol{o} \in M(p)] \qquad (\varphi_{fin})$$

(3) *Moving tokens.* Transitions must be enabled, and tokens are moved by transitions. We encode this as follows:

$$\begin{aligned}\bigwedge_{j=1}^{n} \bigwedge_{l=1}^{L} \mathtt{T}_j = l \rightarrow & \bigwedge_{p \in {}^\bullet t_l \setminus t_l^\bullet} \bigwedge_{\boldsymbol{o} \in \boldsymbol{O}_{\mathtt{color}(p)}} (\mathit{consumed}(p, t_l, j, \boldsymbol{o}) \rightarrow \mathtt{M}_{j-1,p,\boldsymbol{o}} \wedge \neg \mathtt{M}_{j,p,\boldsymbol{o}}) \wedge \\ & \bigwedge_{p \in {}^\bullet t_l \cap t_l^\bullet} \bigwedge_{\boldsymbol{o} \in \boldsymbol{O}_{\mathtt{color}(p)}} (\mathit{consumed}(p, t_l, j, \boldsymbol{o}) \rightarrow \mathtt{M}_{j-1,p,\boldsymbol{o}}) \wedge \\ & \bigwedge_{p \in t_l^\bullet} \bigwedge_{\boldsymbol{o} \in \boldsymbol{O}_{\mathtt{color}(p)}} (\mathit{produced}(p, t_l, j, \boldsymbol{o}) \rightarrow \mathtt{M}_{j,p,\boldsymbol{o}}) \wedge \\ & \mathit{synced}(p, t_l, j) \qquad (\varphi_{move})\end{aligned}$$

where $\mathit{consumed}(p, t, j, \boldsymbol{o})$ expresses that token $\boldsymbol{o}$ is consumed from p in the jth transition which is t, similarly $\mathit{produced}(p, t, j, \boldsymbol{o})$ expresses that token $\boldsymbol{o}$ is produced, and $\mathit{synced}(p, t_l, j)$ ensures that, in the case where the flow from p to t uses an $=$-template inscription, all tokens in p are consumed. Formally, *consumed* is encoded as follows, distinguishing two cases:

- if $F_{in}(p, t) = (v_1, \ldots, v_h)$ is a non-variable flow, let $(k_1, \ldots, k_h)$ be the object indices for t of $v_1, \ldots, v_h$. Then

$$\mathit{consumed}(p, t, j, \boldsymbol{o}) := (\bigwedge_{i=1}^{h} \mathtt{O}_{j,k_i} = \mathit{id}(\boldsymbol{o}_i))$$

  i.e., we demand that every variable used in the transition is instantiated to the respective object in $\boldsymbol{o}$. In this case, we set $\mathit{synced}(p, t_l, j) = \top$.
- if $F_{in}(p, t) = (V_1, \ldots, v_h)$ is a variable flow, suppose without loss of generality that $V_1 \in \mathcal{V}_{list}$. Variable V_1 can be instantiated by multiple objects in a transition firing. This is also reflected by the fact that there are several (but at most K) inscription indices corresponding to instantiations of V_1, say $\ell_1, \ldots, \ell_x$. For k_i as above for $i > 1$, we then set

$$\mathit{consumed}(p, t, j, \boldsymbol{o}) := (\bigwedge_{i=2}^{h} \mathtt{O}_{j,k_i} = \mathit{id}(\boldsymbol{o}_i)) \wedge \bigvee_{i=1}^{x} \mathtt{O}_{j,\ell_i} = \mathit{id}(\boldsymbol{o}_1)$$

If V_1 is a $\subseteq$-template inscription, we set again $synced(p, t_l, j) = \top$. Otherwise, V_1 is a $=$-template inscription, and it must be ensured that all tokens from p are consumed. To this end, we set

$$synced(p, t_l, j) = \sum_{\boldsymbol{o} \in \boldsymbol{O}_{\mathtt{color}(p)}} \mathtt{M}_{j-1,p,\boldsymbol{o}} = \sum_{i=1}^{h} (\mathtt{O}_{j,k_i} \neq 0)$$

i.e., the number of tokens present in p at instant $j-1$ must be equal to the number of objects used to instantiate V_1. (Note that, formally, sums over boolean expressions must be encoded using if-then-else constructs; they are omitted here for readability.)

The shorthand *produced* is encoded similarly as *consumed*, using $F_{out}(t, p)$.

(4) *Tokens that are not moved by transitions remain in their place.*

$$\bigwedge_{j=1}^{n+1} \bigwedge_{p \in P} \bigwedge_{\boldsymbol{o} \in \boldsymbol{O}_{\mathtt{color}(p)}} (\mathtt{M}_{j-1,p,\boldsymbol{o}} \leftrightarrow \mathtt{M}_{j,p,\boldsymbol{o}}) \vee \bigvee_{t_l \in p\bullet} (\mathtt{T}_j = l \wedge consumed(p, t, j, \boldsymbol{o})) \vee \bigvee_{t_l \in \bullet p} (\mathtt{T}_j = l \wedge produced(p, t, j, \boldsymbol{o})) \tag{φ_{rem}}$$

(5) *Transitions use objects of suitable type.* To this end, recall that every transition can use at most K objects, which limits instantiations of template inscriptions. For every transition $t \in T$, we can thus enumerate the objects used by it from 1 to K. However, some of these objects may be unused. We use the shorthand $needed_{t,k}$ to express this: $needed_{t,k} = \top$ if the k-th object is necessary for transition t because it occurs in a simple inscription, and $\bot$ otherwise. Moreover, let $ttype(t, k)$ be the type of the k-th object used by transition t. Finally, we denote by O_σ the subset of objects in O of type σ.

$$\bigwedge_{j=1}^{n} \bigwedge_{l=1}^{L} \mathtt{T}_j = l \rightarrow \bigwedge_{k=1}^{K} \left((\neg[needed_{t_l,k}] \wedge \mathtt{O}_{j,k} = 0) \vee \bigvee_{o \in O_{ttype(t_l,k)}} \mathtt{O}_{j,k} = id(o) \right) \tag{φ_{type}}$$

(6) *Objects that instantiate ν-variables are fresh.* We assume in the following constraint that $tids_\nu$ is the set of all $1 \leq l \leq L$ such that t_l has an outgoing ν-inscription, and that every such t_l has only one outgoing ν-inscription ν_t, and we assume w.l.o.g. that in the enumeration of objects of t, ν_t is the first object. However, the constraint can be easily generalized to more such inscriptions.

$$\bigwedge_{j=1}^{n} \bigwedge_{l \in tids_\nu} \bigwedge_{o \in O_{\mathtt{type}(\nu_t)}} \mathtt{T}_j = l \wedge \mathtt{O}_{j,1} = id(o) \rightarrow \Big(\bigwedge_{p \in P} \bigwedge_{\boldsymbol{o} \in \boldsymbol{O}_{\mathtt{color}(p)}, o \in \boldsymbol{o}} \neg \mathtt{M}_{j-1,p,\boldsymbol{o}} \Big) \tag{φ_{fresh}}$$

(7) *Guards are satisfied.* To that end, we set

$$\bigwedge_{j=1}^{n} \bigwedge_{l=1}^{L} \mathtt{T}_j = l \rightarrow \mathtt{guard}(t_l)(\mathtt{O}_{j,1}, \ldots, \mathtt{O}_{j,K}) \qquad (\varphi_{guard})$$

where $\mathtt{guard}(t_l)(\mathtt{O}_{j,1}, \ldots, \mathtt{O}_{j,K})$ is an instantiation of the guard of t_l with the object variables of instant j, using object indices. Here we assume that aggregation functions have a suitable SMT encoding supported by the solver, which is the case for the common aggregations of summation, maximum, minimum, and average.

Encoding alignment cost. Similar as in [9,5], we encode the cost of an alignment as the edit distance with respect to suitable penalty functions $P_=$, P_M, and P_L. Given a trace graph $T_X = (E_X, D_X)$, let

$$\mathbf{e} = \langle e_1, \ldots, e_m \rangle \qquad (1)$$

be an enumeration of all events in E_X such that $\pi_{time}(e_1) \leq \cdots \leq \pi_{time}(e_m)$. Let the penalty expressions $[P_L]_i$, $[P_M]_j$, and $[P_=]_{i,j}$ be as follows, for all $1 \leq i \leq m$ and $1 \leq j \leq n$:

$$[P_L]_i = |\pi_{obj}(e_i)| \qquad [P_=]_{i,j} = ite(is\_labelled(j, \pi_{act}(e_i)), 0, \infty)$$
$$[P_M]_j = ite(is\_labelled(j, \tau), 0, \Sigma_{k=1}^{K}(\mathtt{O}_{j,k} \neq 0))$$

where $is\_labelled(j, a)$ expresses that the j-the transition has label $a \in \mathcal{A} \cup \{\tau\}$, which can be done by taking $is\_labelled(j, a) := \bigvee_{l \in T_{idx}(a)} \mathtt{T}_j = l$ where $T_{idx}(a)$ is the set of transition indices with label a, i.e., the set of all l with $t_l \in T$ such that $\ell(t_l) = a$.

Using these expressions, one can encode the edit distance as in [9,5]:

$$\begin{aligned} \delta_{0,0} = 0 \qquad \delta_{i+1,0} = [P_L] + \delta_{i,0} \qquad \delta_{0,j+1} = [P_M]_{j+1} + \delta_{0,j} \\ \delta_{i+1,j+1} = \min([P_=]_{i+1,j+1} + \delta_{i,j},\ [P_L] + \delta_{i,j+1},\ [P_M]_{j+1} + \delta_{i+1,j}) \end{aligned} \qquad (\varphi_\delta)$$

Solving. We abbreviate $\varphi_{run} = \varphi_{init} \wedge \varphi_{fin} \wedge \varphi_{move} \wedge \varphi_{rem} \wedge \varphi_{type} \wedge \varphi_{fresh} \wedge \varphi_{guard}$ and use an SMT solver to obtain a satisfying assignment α for the following constrained optimization problem:

$$\varphi_{run} \wedge \varphi_\delta \quad \text{minimizing} \quad \delta_{m,n} \qquad (\Phi)$$

Decoding. From an assignment α satisfying (Φ), we next define a run ρ_α and an alignment Γ_α. First, we note the following: From Lemma 1, we can obtain a number M such that M is the maximal number of objects used to instantiate a list variable in the model run and alignment. By convention, we may assume that in the enumeration of objects used in the jth transition firing, $\mathtt{O}_{j,|O|-M+1}, \ldots, \mathtt{O}_{j,|O|}$ are those instantiating a list variable, if there is a list variable in $vars_{in}(t_{\alpha(\mathtt{T}_j)}) \cup vars_{out}(t_{\alpha(\mathtt{T}_j)})$.

We assume the set of transitions $T = \{t_1, \ldots, t_L\}$ is ordered as $t_1, \ldots, t_L$ in some arbitrary but fixed way that was already used for the encoding.

Definition 12 (Decoded run). *For α satisfying (Φ), let the* decoded process run *be $\rho_\alpha = \langle f_1, \ldots, f_n \rangle$ such that for all $1 \leq j \leq n$, $f_j = (\widehat{t}_j, b_j)$, where $\widehat{t}_j = t_{\alpha(\mathtt{T}_j)}$ and b_j is defined as follows: Assuming that $vars_{in}(t_{\alpha(\mathtt{T}_j)}) \cup vars_{out}(t_{\alpha(\mathtt{T}_j)})$ is ordered as $v_1, \ldots, v_k$ in an arbitrary but fixed way that was already considered for the encoding, we set $b_j(v_i) = \alpha(\mathtt{O}_{j,i})$ if $v_i \in \mathcal{V}$, and $b_j(v_i) = [O_{\alpha(\mathtt{O}_{j,|O|-M+1})}, \ldots, O_{\alpha(\mathtt{O}_{j,|O|-M+z})}]$ if $v_i \in \mathcal{V}$, where $0 \leq z < M$ is maximal such that $\alpha(\mathtt{O}_{j,|O|-M+z}) \neq 0$.*

At this point, ρ_α is actually just a sequence; we will show below that it is indeed a process run of $\mathcal{N}$. Next, given a satisfying assignment α for (Φ), we define an alignment of the log trace T_X and the process run ρ_α.

Definition 13 (Decoded alignment). *For α satisfying (Φ), $\rho_\alpha = \langle f_1, \ldots, f_n \rangle$ as defined above, and $\mathbf{e}$ as in (1), consider the sequence of moves $\gamma_{i,j}$ recursively defined as follows:*

$$\gamma_{0,0} = \epsilon \qquad \gamma_{i+1,0} = \gamma_{i,0} \cdot \langle e_{i+1}, \gg \rangle \qquad \gamma_{0,j+1} = \gamma_{0,j} \cdot \langle \gg, f_{j+1} \rangle$$

$$\gamma_{i+1,j+1} = \begin{cases} \gamma_{i,j+1} \cdot \langle e_{i+1}, \gg \rangle & \text{if } \alpha(\delta_{i+1,j+1}) = \alpha([P_L] + \delta_{i,j+1}) \\ \gamma_{i+1,j} \cdot \langle \gg, f_{j+1} \rangle & \text{if otherwise } \alpha(\delta_{i+1,j+1}) = \alpha([P_M]_{j+1} + \delta_{i+1,j}) \\ \gamma_{i,j} \cdot \langle e_{i+1}, f_{j+1} \rangle & \text{otherwise} \end{cases}$$

Given $\gamma_{i,j}$, we define a graph $\Gamma(\gamma_{i,j}) = \langle C, B \rangle$ of moves as follows: the node set C consists of all moves in $\gamma_{i,j}$, and there is an edge $\langle \langle q, r \rangle, \langle q', r' \rangle \rangle \in B$ if either $q \neq \gg$, $q' \neq \gg$ and there is an edge $q \to q'$ in T_X, or if $r \neq \gg$, $r' \neq \gg$, $r = f_h$, and $r' = f_{h+1}$ for some h with $1 \leq h < n$. Finally, we define the decoded alignment as $\Gamma(\alpha) := \Gamma(\gamma_{m,n})$.

In fact, as defined, $\Gamma(\alpha)$ is just a graph of moves, it yet has to be shown that it is a proper alignment. This will be done in the next section.

Correctness. In the remainder of this section, we will prove that ρ_α is indeed a run, and $\Gamma(\alpha)$ is an alignment of T_X and ρ_α. We first show the former:

Lemma 2. *Let $\mathcal{N}$ be a DOPID, T_X a log trace and α a solution to (Φ). Then ρ_α is a run of $\mathcal{N}$.*

Proof. We define a sequence of markings $M_0, \ldots, M_n$. Let M_j, $0 \leq j \leq n$, be the marking such that $M_j(p) = \{\boldsymbol{o} \mid \boldsymbol{o} \in \boldsymbol{O}_{\mathtt{color}(p)} \text{ and } \alpha(\mathtt{M}_{j,p,\boldsymbol{o}}) = \top\}$. Then, we can show by induction on j that for the process run $\rho_j = \langle f_1, \ldots, f_j \rangle$ it holds that $M_0 \xrightarrow{\rho_j} M_j$.

Base case. If $n = 0$, then ρ_0 is empty, so the statement is trivial.
Inductive step. Consider $\rho_{j+1} = \langle f_1, \ldots, f_{j+1} \rangle$ and suppose that for the prefix $\rho' = \langle f_1, \ldots, f_j \rangle$ it holds that $M_0 \xrightarrow{\rho'} M_j$. We have $f_{j+1} = (\widehat{t}, b)$ and $\widehat{t} = t_i$ for some i such that $1 \leq i \leq |T|$ with $\alpha(\mathtt{T}_j) = i$. First, we note that b is a valid binding: as α satisfies (φ_{type}), it assigns a non-zero value to all $\mathtt{O}_{j,k}$ such that $v_k \in vars_{in}(t_i) \cup vars_{out}(t_i)$ that are not of list type (and hence needed), and by (φ_{type}), the unique object o with $id(o) = \alpha(\mathtt{O}_{j,k})$ has the

type of v_k. Similarly, b assigns a list of objects of correct type to a variable in $(vars_{in}(t_i) \cup vars_{out}(t_i)) \cap \mathcal{V}_{list}$, if such a variable exists. Moreover, (φ_{fresh}) ensures that variables in $(vars_{in}(t_i) \cup vars_{out}(t_i)) \cap \Upsilon$ are instantiated with objects that did not occur in M_j, and (φ_{guard}) ensures taht the guard is satisfied.
Since α is a solution to (Φ), it satisfies (φ_{move}), so that t_i is enabled in M_n. Moreover, the distinction between $\subseteq$- and $=$-inscriptions is taken care of by the *synced* predicate. As α satisfies (φ_{rem}), the new marking M_{j+1} contains only either tokens that were produced by t_i, or tokens that were not affected by t_i. Thus, $M_j \xrightarrow{f_{j+1}} M_{j+1}$, which concludes the induction proof.

Finally, as α satisfies (φ_{init}) and (φ_{fin}), it must be that $M_0 = M_I$ and the last marking must be final, so ρ_α is a run of $\mathcal{N}$. □

Theorem 1. *Given a DOPID $\mathcal{N}$, trace graph T_X, and satisfying assignment α to (Φ), $\Gamma(\alpha)$ is an optimal alignment of T_X and the run ρ_α with cost $\alpha(\delta_{m,n})$.*

Proof. By Lem. 2, ρ_α is a run of $\mathcal{N}$. We first note that $[P_=]$, $[P_L]$, and $[P_M]$ are correct encodings of $P_=$, P_L, and P_M from Def. 10, respectively. For P_L this is clear. For $P_=$, $is\_labelled(j, a)$ is true iff the value of $\mathtt{T}_j$ corresponds to a transition that is labeled a. If the labels match, cost 0 is returned, otherwise ∞. For P_M, the case distinction returns cost 0 if the jth transition is silent; otherwise, the expression $\Sigma_{k=1}^{K} ite(\mathtt{O}_{j,k} \neq 0, 1, 0)$ counts the number of objects involved in the model step, using the convention that if fewer than k objects are involved in the jth transition then $\mathtt{O}_{j,k}$ is assigned 0.

Now, let $d_{i,j} = \alpha(\delta_{i,j})$, for all i, j such that $0 \leq i \leq m$ and $0 \leq j \leq n$. Let again $\mathbf{e} = \langle e_1, \ldots, e_m \rangle$ be the sequence ordering the nodes in T_X as in (1). Let $T_X|_i$ be the restriction of T_X to the node set $\{e_1, \ldots, e_i\}$. We show the stronger statement that $\Gamma(\gamma_{i,j})$ is an optimal alignment of $T_X|_i$ and $\rho_\alpha|_j$ with cost $d_{i,j}$, by induction on (i, j).

Base case. If $i = j = 0$, then $\gamma_{i,j}$ is the trivial, empty alignment of an empty log trace and an empty process run, which is clearly optimal with cost $d_{i,j} = 0$, as defined in (φ_δ).

Step case. If $i = 0$ and $j > 0$, then $\gamma_{0,j}$ is a sequence of model moves $\gamma_{0,j} = \langle (\gg, f_1), \ldots, (\gg, f_j) \rangle$ according to Def. 13. Consequently, $\Gamma(\alpha) = \Gamma(\gamma_{0,j})$ has edges $(\gg, f_h), \ldots, (\gg, f_{h+1})$ for all h, $1 \leq h < j$, which is a valid and optimal alignment of the empty log trace and ρ_α. By Def. 10, the cost of $\Gamma(\alpha)$ is the number of objects involved in non-silent transitions of $f_1, \ldots, f_j$, which coincides with $\alpha([P_M]_1 + \cdots + [P_M]_j)$, as stipulated in (φ_δ).

Step case. If $j = 0$ and $i > 0$, then $\gamma_{i,0}$ is a sequence of log moves $\gamma_{i,0} = \langle (e_1, \gg), \ldots, (e_j, \gg) \rangle$ according to Def. 13. Thus, $\Gamma(\alpha) = \Gamma(\gamma_{i,0})$ is a graph whose log projection coincides by definition with $T_X|_i$. By Def. 10, the cost of $\Gamma(\alpha)$ is the number of objects involved in $e_1, \ldots, e_i$, which coincides with $\alpha([P_L]_1 + \cdots + [P_L]_j)$, as stipulated in (φ_δ).

Step case. If $i > 0$ and $j > 0$, $d_{i,j}$ must be the minimum of $\alpha([P_=]_{i,j}) + d_{i-1,j-1}$, $\alpha([P_L]) + d_{i-1,j}$, and $\alpha([P_M]_j) + d_{i,j-1}$. We can distinguish three cases:

- Suppose $d_{i,j} = \alpha([P_L]_i) + d_{i-1,j}$. By Def. 13, we have $\gamma_{i,j} = \gamma_{i-1,j} \cdot \langle e_i, \gg \rangle$. Thus, $\Gamma(\gamma_{i,j})$ extends $\Gamma(\gamma_{i-1,j})$ by a node $\langle e_i, \gg \rangle$, and edges to this node as induced by $T_X|_i$. By the induction hypothesis, $\Gamma(\gamma_{i-1,j})$ is a valid and optimal alignment of $T_X|_{i-1}$ and $\rho_\alpha|_j$ with cost $d_{i-1,j}$. Thus $\Gamma(\gamma_{i,j})$ is a valid alignment of $T_X|_i$ and $\rho_\alpha|_j$, because the log projection coincides with $T_X|_i$ by definition. By minimality of the definition of $d_{i,j}$, also $\Gamma(\gamma_{i,j})$ is optimal.
- Suppose $d_{i,j} = \alpha([P_M]_j) + d_{i,j-1}$. By Def. 13, we have $\gamma_{i,j} = \gamma_{i,j-1} \cdot \langle \gg, f_j \rangle$. By the induction hypothesis, $\Gamma(\gamma_{i,j-1})$ is a valid and optimal alignment of $T_X|_i$ and $\rho_\alpha|_{j-1}$ with cost $d_{i,j-1}$. Thus, $\Gamma(\gamma_{i,j-1})$ must have a node $\langle r, f_{j-1} \rangle$, for some r. The graph $\Gamma(\gamma_{i,j})$ extends $\Gamma(\gamma_{i,j-1})$ by a node $\langle \gg, f_j \rangle$, and an edge $\langle r, f_{j-1} \rangle \to \langle \gg, f_j \rangle$. Thus, $\Gamma(\gamma_{i,j})$ is a valid alignment for $T_X|_i$ and $\rho_\alpha|_j$, and by minimality it is also optimal.
- Let $d_{i,j} = \alpha([P_=]_{i,j}) + d_{i-1,j-1}$. By Def. 13, we have $\gamma_{i,j} = \gamma_{i-1,j-1} \cdot \langle e_i, f_j \rangle$. By the induction hypothesis, $\Gamma(\gamma_{i-1,j-1})$ is an optimal alignment of $T_X|_{i-1}$ and $\rho_\alpha|_{j-1}$ with cost $d_{i-1,j-1}$. In particular, $\Gamma(\gamma_{i-1,j-1})$ must have a node $\langle r, f_{j-1} \rangle$, for some r. The graph $\Gamma(\gamma_{i,j})$ extends $\Gamma(\gamma_{i-1,j-1})$ by a node $\langle e_1, f_j \rangle$, an edge $\langle r, f_{j-1} \rangle \to \langle e_i, f_j \rangle$, as well as edges to $\langle e_1, f_j \rangle$ as induced by $T_X|_i$. Thus $\Gamma(\gamma_{i,j})$ is a valid alignment of $T_X|_i$ and $\rho_\alpha|_j$, because the log projection coincides with $T_X|_i$ by definition, and the model projection has the required additional edge. By minimality of the definition of $d_{i,j}$, also $\Gamma(\gamma_{i,j})$ is optimal.

For the case $i = m$ and $j = n$, we obtain that $\Gamma(\alpha) = \Gamma(\gamma_{m,n})$ is an optimal alignment of T_X and ρ_α with cost $d_{m,n} = \alpha(\delta_{m,n})$. □