Journal Européen des Systèmes Automatisés
Vol. 59, No. 10, October, 2026, pp. 3067-3077
Journal homepage: http://iieta.org/journals/jesa

The Object-Centric Online Diagnosis of a Telecommunications Order Fulfilment Process

Nur Ichsan Utama[1], Hanung Nindito Prasetyo[2], Gede Agung Ary Wisudiawan[3], Angelina Prima Kurniati[3*]

[1] Information System Study Program, School of Industrial Engineering, Telkom University, Bandung 40257, Indonesia
[2] Information System Diploma Study Program, School of Applied Science, Telkom University, Bandung 40257, Indonesia
[3] Informatics Study Program, School of Computing, Telkom University, Bandung 40257, Indonesia

Corresponding Author Email: angelina@telkomuniversity.ac.id

Copyright: ©2026 The authors. This article is published by IIETA and is licensed under the CC BY 4.0 license (http://creativecommons.org/licenses/by/4.0/).

https://doi.org/10.18280/jesa.591025

Received: 22 July 2026
Revised: 14 September 2026
Accepted: 21 October 2026
Available online: 31 October 2026

Keywords:
object-centric process mining, digital twin, discrete event systems, online conformance checking, fault diagnosis, telecommunications

ABSTRACT

Order fulfilment in a telecommunications operator is a discrete event system in which several kinds of objects evolve at once. A customer order spawns work orders, which raise fault tickets that a separate system resolves on a different clock. Diagnosis is nevertheless performed one case notion at a time, after the fact, at an unmeasured cost. This paper reports an object-centric digital twin that performs the diagnosis online, meaning per event as the stream arrives; it is evaluated by replaying historical logs and was not connected to a production system. An object-centric event log is constructed from four industrial exports, recovering a work order to ticket relation that an earlier internal analysis reported as unavailable because the documented key is empty in all five million rows. Routing it through the customer order links 539,248 pairs over 2.94 million events. Object-centric directly-follows graphs (OC-DFGs), Petri nets and Business Process Model and Notation (BPMN) lanes are discovered from it, and the customer order is shown to span all ten activities where a work order spans seven. Flattening onto the work order, the case notion every prior analysis used, discards half of all events. Conformance per object type and operator-defined rules run in one pass over per-object state, in a compiled engine verified against a reference implementation. Four deviation classes are introduced that depend on two objects at once. On one million events, the twin reports 13,580 work orders reaching a completing activity while a fault ticket against the same order is still open, a finding neither lane shows alone; a validation sample finds 70.8% of reported deviations genuine. Object-centricity costs 81.0 nanoseconds per event-to-object relation against 97.1 nanoseconds per event for the single-case-notion path, medians of ten repetitions, so the slowdown is the additional objects rather than structural overhead.

1. INTRODUCTION

Order fulfilment at a national telecommunications operator is a high-volume, low-margin activity whose data volume is itself a recognised analytical challenge [1]. The operator studied here serves more than ten million customers, and one extraction window records five million work order status transitions. When an order stalls, the cost is paid in recovery effort and in a service delivered late.

Process mining [2, 3] has been applied to this class of problem, and the present authors have used it descriptively in other domains [4, 5]. This operator's fault data has been examined once before, in an unpublished internal analysis that Section 3.2 revisits. Both are retrospective: deviations are reported after the affected order has failed.

Three problems stand between that and a system operations staff can use. The first is reference: a mined model describes what does happen, failures included, while a model transcribed from the standard operating procedure describes only what the document chose to draw. The second is cost: rules must be evaluated per event against per-object state at the rate the process generates events. The third, and the subject of this paper, is case notion. Every analysis of this process, the authors' own included, treats the work order as the unit of analysis, a modelling decision whose consequences have not been measured. A customer order spawns several concurrent work orders and raises fault tickets that open and close on an unrelated clock, and choosing any one of these as the case discards the others.

The contributions are:

1. An object-centric event log for an industrial fulfilment process, and the recovery of a work order-to-ticket relationship that an earlier internal analysis of this data reported as unavailable. Its coverage is measured, not assumed, and the shortfall is surfaced by the runtime as a deviation class rather than resolved silently during preparation.

2. A measured object-centricity profile. Convergence, divergence, and deficiency are computed for every candidate case notion, and the cost of flattening onto the conventional one is quantified.

3. An object-centric digital twin in which conformance against a per-type process model and evaluation of operator-defined rules run in a single pass over per-object state, adding four deviation classes that no single-case-notion checker can

express, with the cost of object-centricity measured against the same binary running its single-case-notion path.

Rule generation by a language model is a component of that system and not a fourth contribution. It is measured in Section 4.7, and the result argues for the validation gate around it rather than for the generator.

2. RELATED WORK

Object-centric process mining. Flattening an event log onto one case notion introduces convergence, divergence, and deficiency [6]. The object-centric event log (OCEL) standard provides an interchange format [7], object-centric Petri nets (OCPNs) are a discovery target [8], and precision and fitness have been redefined for the setting [9]. Extraction from relational sources [10] and from unstructured fields [11] supplies the relations between object types. Conformance has been addressed offline [12], and operational monitoring has an object-centric formulation [13]. Conformance has since been formulated through object-centric alignments [14] and on object-centric causal nets [15], with tool support for discovery and conformance [16] and for performance analysis across interacting objects [17].

Conformance checking on streams. Alignment-based conformance [18] gives optimal-cost explanations but requires a search over repair paths, so online variants relax it: behavioural patterns replace alignments [19], prefix-alignments are maintained incrementally [20], and the area has a dedicated survey [21]. Recent work computes exact prefix-alignments by incremental state-space expansion [22], orients a running case probabilistically before judging it [23], checks conformance under limits on data retention [24], and detects deviations in event streams with variational autoencoders [25]. Tracking a process whose log is incomplete has also been studied [26]. The checker of Section 3.4 is an incremental token-replay approximation in this tradition.

AI-augmented and prescriptive process management. Research agendas place AI inside process execution rather than only reporting on it [27-29]. Predictive monitoring has been surveyed [30] with sustained attention to explainability [31, 32], prescriptive monitoring frames intervention as an alarm-based decision [33] or an explicit policy [34], and object-centric information improves prediction [35]. These lines assume the events needed to act are available, which this work treats as something to be measured.

Stream processing and digital twins. Complex event processing evaluates standing queries over unbounded streams [36], and RETE [37] amortises joins across many facts in a large working memory, a different workload from bounded per-object checks. The digital twin originates in product lifecycle management [38], is established in manufacturing [39] and is offered as a service [40]. For business processes, a digital twin of an organization has been realised with action-oriented process mining that turns constraint violations into actions [41], simulation-based twins monitor patient pathways in near real time [42], and digital process twins have been classified [43]. Unlike a physical twin, a process twin can see only what the information systems already record.

Language models generating executable artefacts. Language models generate code [44] and Structured Query Language (SQL) [45] with useful reliability; their capability on process mining tasks has begun to be measured [46, 47], and conversational process-management interfaces have been proposed [48]. An executable rule differs from an answer about a log because it fires actions autonomously.

Where this work sits. These areas supply three of the four things an object-centric online diagnosis needs and leave the fourth open. Object-centric process mining supplies the representation and the discovery targets [6-9], but its conformance and monitoring formulations compute over a stored log [12, 13]. Online conformance checking supplies per-event evaluation [19-21], but on a single-case notion, so a constraint relating two object types cannot be stated in it. AI-augmented process management supplies the argument for acting rather than reporting [27-29], and predictive and prescriptive monitoring supply the decision layer [30, 33, 34], both assuming the events needed are already available. What none supplies is a checker that evaluates constraints across object types per event, at arrival, from state bounded in advance. The closest work is Park and van der Aalst [13], and the difference is specific rather than one of degree. Their system defines object-centric monitoring measures and computes them over a stored log, recomputing a measure when an analyst asks for it, so nothing in that design must keep pace with an arrival rate and no measure need be decidable from state already held. The twin described here evaluates constraints per event from per-object state bounded in advance: every inter-type constraint in Section 3.4 is answerable in constant time from counters maintained on the write path, which is why a count over related objects is one array read and not a traversal. Three things follow that a stored-log formulation does not have to supply. Constraints spanning two lifecycles are checked as events arrive rather than after the fact. The checker is verified against an independent implementation of the same bytecode before any timing is reported. And the cost of object-centricity is itself measured, in nanoseconds per event-to-object relation, a quantity an offline formulation never needs to produce.

Two further works are close enough to compare directly. Object-centric alignments [14] give optimal-cost explanations of object-centric deviation, which the checker described here does not: it trades optimality for constant-time evaluation per event, as token-replay approximations do in the single-case setting [19, 20], and reports approximate classes rather than minimal repair costs. The digital twin of an organization [41] shares the aim of acting on constraint violations rather than reporting them, and the twin here can be read as one realisation of that aim for a process whose constraints span object types. What it adds is that the constraints are evaluated from per-object state bounded in advance, so their cost can be measured and verified before any action depends on them.

3. METHODOLOGY

The work follows a design-oriented approach: an artefact is built for a stated operational problem and evaluated on the data and at the scale it is meant to serve, with correctness established by differential verification before any performance claim is made. Implementation details that support reproducibility but not the diagnostic contribution, including the compilation scheme, the runtime state layout and the verification gate, are given in the Supplementary Material.

3.1 System architecture

Figure 1 shows the system. An event source yields events,

each carrying a timestamp, an activity, and the set of objects it relates to; a replay driver reading a prepared columnar log and a live push endpoint both satisfy it, and the runtime cannot distinguish them. The object-centric log feeds two consumers. Object-centric discovery produces the models of Section 3.3 using pm4py, for analysts. A per-type model representation, holding one lifecycle model per object type together with the constraints between types, is compiled once, offline, for the runtime. For each event, the runtime resolves the objects the event relates to, checks each against its own type's model, evaluates operator rules with the resulting deviation flags available as operands, and places any firing on a queue from which a separate dispatcher delivers it.

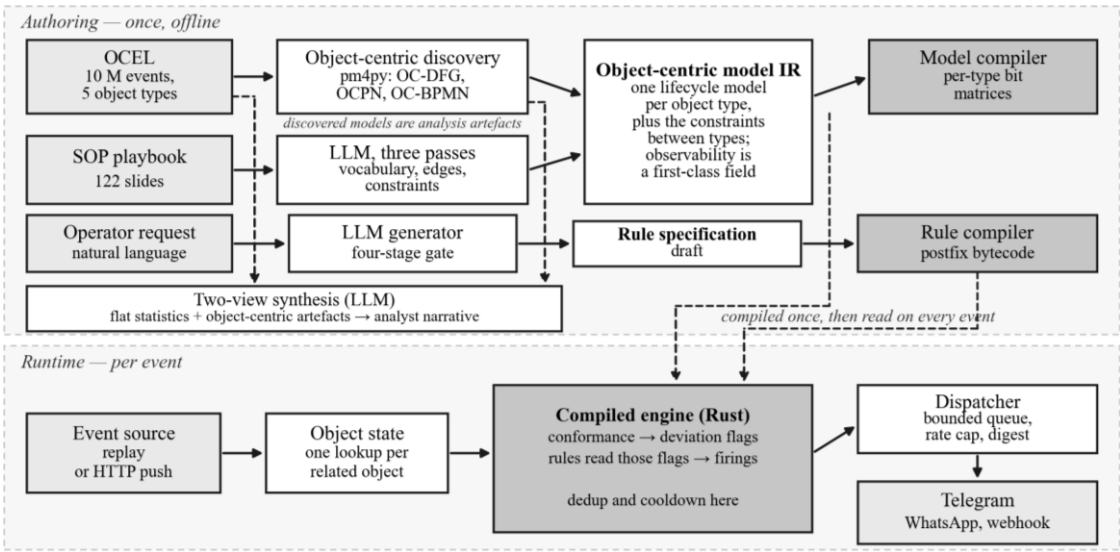


Figure 1. System architecture

Note: Models and rules are compiled once, offline, and read on every event at runtime. The engine's measured cost ends where a firing is enqueued. The two-view synthesis is an offline analysis path and reaches no part of the runtime

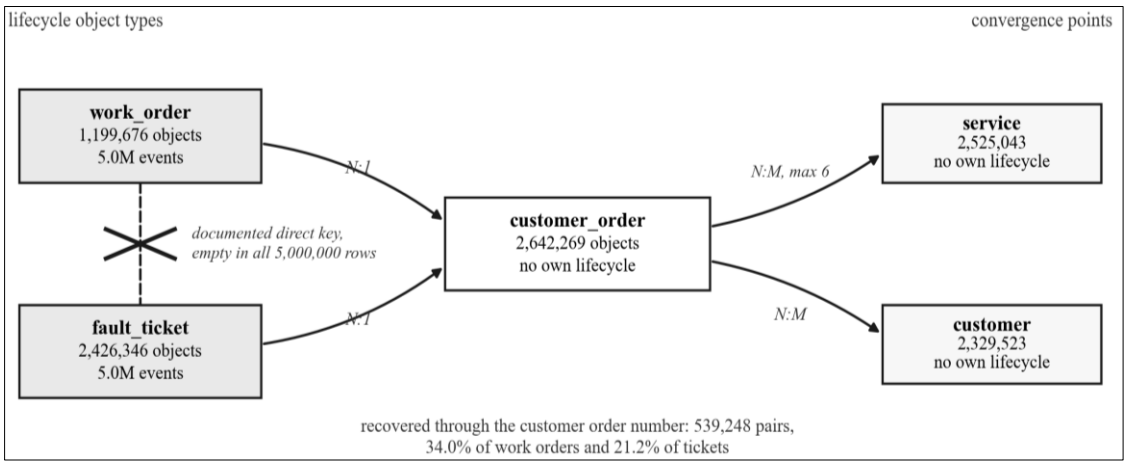


Figure 2. The object-centric data model

Note: The relation the operator's data dictionary documents between work orders and fault tickets is empty in every row. The two lifecycles are reconnected through the customer order they share

3.2 Object-centric event log construction

The operator supplied four exports: a work order status history, a fault ticket history, a work order master table, and a spreadsheet whose filename describes it as a fault data summary. The last is not a summary. Its embedded query sheet shows a full extract of the incident table named in the operator's data dictionary, holding 5,046,821 rows, and it is the only export carrying a work order number and a ticket identifier on the same row.

The internal analysis referred to in Section 1 reported that the work order and fault streams cannot be joined because the documented foreign key is empty. A full scan confirms the column is empty in all 5,000,000 rows, but the conclusion drawn from it was too strong. Two bridges exist, and both were measured over the complete populations instead of being sampled. The direct bridge uses the incident table's work order number. The indirect bridge uses the customer order number, which appears in the work order master and is embedded in the machine-generated description text of every fault record, that text being a recognised source of object relations [11]. The order identifier has more than one syntactic form on both sides, so the alignment matters: truncating both sides at the first underscore matches 2,656,365 keys, truncating neither matches 2,460,581, and either mixed alignment matches none. An earlier preprocessing script truncated one side only, which is the immediate cause of the relation appearing dead.

Five object types result, and Figure 2 gives the object-

centric data model. Work order and fault ticket have lifecycles of their own. Customer order, service, and customer are referenced by events, but no export records their state changes, and the distinction is recorded in the model so that no consumer mistakes five identifiers for five mineable processes. The result is written as an OCEL 2.0 log, so it is interchangeable with standard tooling.

3.3 Object-centric process models

Object-centric process mining has established model formalisms, and this work uses them instead of substituting its own. Three are discovered from the training split with pm4py, and their sizes are reported in Section 4.2. The object-centric data model names the object types, their identifiers, and the relations between them, with measured cardinalities; it is the schema in Figure 2. The object-centric directly-follows graph (OC-DFG) is one graph over the shared activity set, with a separate directly-follows relation for each object type, and it is the representation in which the interleaving between lifecycles becomes visible. The OCPN [8] gives each object type a typed subnet over a shared transition set, with variable arcs where a transition consumes or produces several objects of one type. An object-centric Business Process Model and Notation (BPMN) rendering is also produced, one lane per object type, because that is the form operations staff read most easily.

These are discovery artefacts, and the runtime does not execute them. For checking, each object type keeps its own lifecycle model over its own activity alphabet, and constraints spanning two types are held separately. The model records which activities the stream can and cannot show: two activities named by the standard operating procedure, W3 and W4, are emitted in a different lane and never occur in the work order stream, so declaring them unobservable reduces the nine-node documented path to the seven-node observable projection W1 → W2 → W5 → W6 → W7 → W8 → W9, without which conformance against the documented path would score near zero for every case. The reported per-event costs in Section 4.6 therefore measure this compiled per-type checking, not the execution of a Petri net; the observable projection is drawn in the Supplementary Material.

3.4 Online object-centric conformance checking

Each event is classified in constant time. Seven classes concern a single object and are listed in Table 1.

Table 1. Deviation classes raised for a single object

	Deviation	Test
D1	unexpected transition	edge absent from that type's model
D2	skipped required	required predecessors not yet seen
D3	latency violation	transition exceeded its duration envelope
D4	loop exceeded	activity repeated beyond its bound
D5	silent skip	D1, but reachable through unobservable activities
D6	absence within deadline	raised by the timer wheel, not by an event
D7	unmodelled activity	the activity is outside that type's alphabet

D5 makes partial observability measurable. The ratio states how much apparent non-conformance is in fact unobservability.

$$\rho_{skip} = \frac{|D5|}{|D1| + |D5|} \tag{1}$$

Four further classes depend on what an event says about two objects at once, and none can be derived from D1 to D7, however the checker is configured. They are given in Table 2.

Table 2. Deviation classes that span two objects

	Deviation	Test	Cost
D8	orphan event	no object of a type the model requires	one length check
D9	parent state forbidden	an object entered an activity a related object's state forbids	one mask test per related object
D10	sibling stall	a sibling occupies an activity the model declares unique per parent	one counter read
D11	convergence conflict	the event bound more objects of a type than the model permits	one pass over the fan-out

The four classes are not arbitrary. Each names a way two objects can disagree that the operator's standard operating procedure already forbids for a single object, lifted to a pair. D9 and D10 were derived from the completion and escalation rules in that procedure. D8 and D11 follow instead from the object-centric data model of Section 3.2, and are included because a log can violate its own schema. None was proposed by the operator, and confirming them against operational practice is left to future work. D9 carries the operational finding. A work order and a fault ticket never appear in the same event and see each other only through the customer order they share, so the check takes one hop to that order and at most a compile-time bound from there.

All checks for an event run in one pass over the state of the objects it relates to. Counts over related objects are maintained as those objects move, so a cross-object check reads a counter rather than traversing children, and a bound on the number of related objects is fixed at compile time from the measured degree of each type. The checker resynchronises optimistically, so one deviating event never cascades, and an object that has left its model is reported once. The state layout and the compilation scheme are described in the Supplementary Material.

3.5 Rule generation and validation

This component is supporting infrastructure: it lets an operator express a constraint without writing code, but nothing in the twin depends on it, and no claim in this paper rests on its output. A rule has a trigger, a condition, an optional deadline, a deduplication policy, and a list of actions, and it is specified in JavaScript Object Notation (JSON) validated against a schema. Object-centricity adds operands that count related objects of a named type in named states or read a field of a related object, for example:

count(related work_order IN_STATE [W10]) > 2

evaluated against a customer order. A language model produces the specification from a request, with a prompt

carrying only the closed vocabulary of the stream: observed activities, attribute domains, object types and relations, deviation flags and action schemas. Identifying attributes are described by cardinality alone and never enumerated, because the export contains staff names, customer names, telephone numbers and service addresses. Every generated rule then passes the four stages of Table 3, receives at most one repair round, and is created as a draft that must be simulated over history and activated by a person. The same model also writes an offline narrative combining the flattened statistics with the object-centric results, reported in Section 4.7.

Table 3. The four-stage validation gate applied to every generated rule

Stage	Check
S1	The response parses as JSON
S2	It satisfies the rule schema
S3	Every activity, attribute, value, object type, and action exists
S4	It compiles and passes static checks for rules that can never fire

Note: JSON: JavaScript Object Notation

3.6 Experimental setup and evaluation protocol

The corpora are historical extractions, replayed at controlled rates through the same ingestion interface a live feed would use. No connection to a production system was established, and no action was applied to a production work order, so throughput and latency characterise the engine under controlled load and are not measurements of the operator's live load. Timestamps come from the record creation column. Thresholds and envelopes are derived on the earlier 60% of the timeline only, split temporally rather than at random, so that an object's later behaviour cannot inform the bound used to judge it.

Labels are anonymised at the operator's request. Every published label is a code: W1 to W15 for the work-order lifecycle and T1 to T3 for the fault-ticket lifecycle, with W1 the initial approval state, W9 the normal completion, W10 the failure state, and T1, T2 and T3 the open, resolved and closed states of a ticket. The mapping is applied when the log is built, so that models drawn by a third-party library carry only the codes, and it is not published.

Rule generation and the narrative use Qwen3.8-27B through a hosted endpoint at temperature 0.1, with its full 262,144-token context and no cap on completion length. Measurements were taken on an Apple M5 Pro with 15 cores and 24 GB of memory, on Python 3.11 and Rust 1.97. Object-centric discovery and the contextual baseline of Section 4.6 use pm4py 2.7.19. Before any timing, the compiled engine and an independent reference implementation of the same programs process the same stream and must produce identical firings and identical per-object deviation vectors, and every deviation class must be raised at least once on the stream or on a fixture; the gate is described in the Supplementary Material.

4. RESULTS AND DISCUSSION

4.1 The object-centric event log and its coverage

The constructed log holds 10,000,000 events and 38,787,073 event-to-object relations over five object types, a mean fan-out of 3.88 objects per event. Table 4 gives the object counts.

Table 4. Object types in the constructed log

Object Type	Lifecycle	Objects	Relations
Work order	yes	1,199,676	5,000,000
Fault ticket	yes	2,426,346	5,000,000
Customer order	no	2,642,269	9,999,732
Service	no	2,525,043	9,554,169
Customer	no	2,329,523	9,233,172

Table 5 reports the two bridges. The customer order route reaches materially more than the direct key, and 138,218 tickets are reachable only that way.

Table 5. Coverage of the work order to fault ticket relation

Path	Pairs	Tickets	Work Orders
Direct incident key	375,257	15.5%	24.1%
Via customer order	539,137	21.2%	34.0%
Union	539,248	21.2%	34.0%

The linked portion carries 2,942,026 events. The pattern that matters is not the totals but their overlap: of 2,642,269 customer orders, only 392,610 are touched by both lanes, and that small set is where an inter-type constraint can fire at all.

The remaining coverage gap has two measured causes, neither of them the join. The incident table's work order number is 86% to 99% filled for the first five months of the window and then falls to zero, because the upstream system stopped writing the key. Separately, both event exports stop at exactly 5,000,000 rows, so the status history covers 1,199,676 of 5,240,640 work orders, and many tickets cannot reach a work order whose events were never extracted. An uncapped extract has been requested from the operator.

The relation also carries signal and not only structure. Work orders with at least one fault ticket fail substantially more often than those without. W10 occurs on 6.95% of their events against 4.04%, and W13 on 3.58% against 2.28%, while W9 is lower at 10.44% against 13.91%. Neither lane shows this alone, which is the practical argument for constructing the relation at all.

4.2 The discovered object-centric models

Table 6 gives the sizes of the three models discovered from the training split. Discovery is not the expensive part: the directly-follows graph takes 6.6 seconds and the Petri net 4.4 seconds over 100,000 events and 395,318 relations.

The row that matters is the customer order: it spans all ten activities and 90 directly-follows edges, against seven and 41 for a work order, because it is the object through which the two lifecycles meet. Its most frequent edges include both W1 → W2, a work-order transition, and T1 → T2, a ticket transition. No flattening onto either lifecycle can show that, and it is what Figure 3 renders.

4.3 How object-centric is this process?

Table 7 gives the profile for every candidate case notion. Support is the share of events that relate to at least one object of that type, and deficiency is its complement. The convergence factor is the event duplication a flattened log would incur, and the convergence rate is the share of events

that would have to be duplicated at all. Divergence is measured in the attributed form, meaning the share of repeated activities explained by distinct objects of another type being collapsed together rather than by a genuine loop.

Table 6. Object-centric models discovered from the training split

Object Type	OC-DFG Activities	OC-DFG Edges	OCPN Places	OCPN Transitions	OCPN Arcs	BPMN Nodes	BPMN Flows
work_order	7	41	14	17	40	21	30
fault_ticket	3	9	8	8	20	13	17
customer order	10	90	20	23	58	32	47
service	10	94	15	19	48	27	41
customer	10	100	22	22	62	34	52
Total	10 distinct	334	79	89	228	127	187

Note: OC-DFG: object-centric directly-follows graph; OCPN: object-centric Petri net; BPMN: Business Process Model and Notation

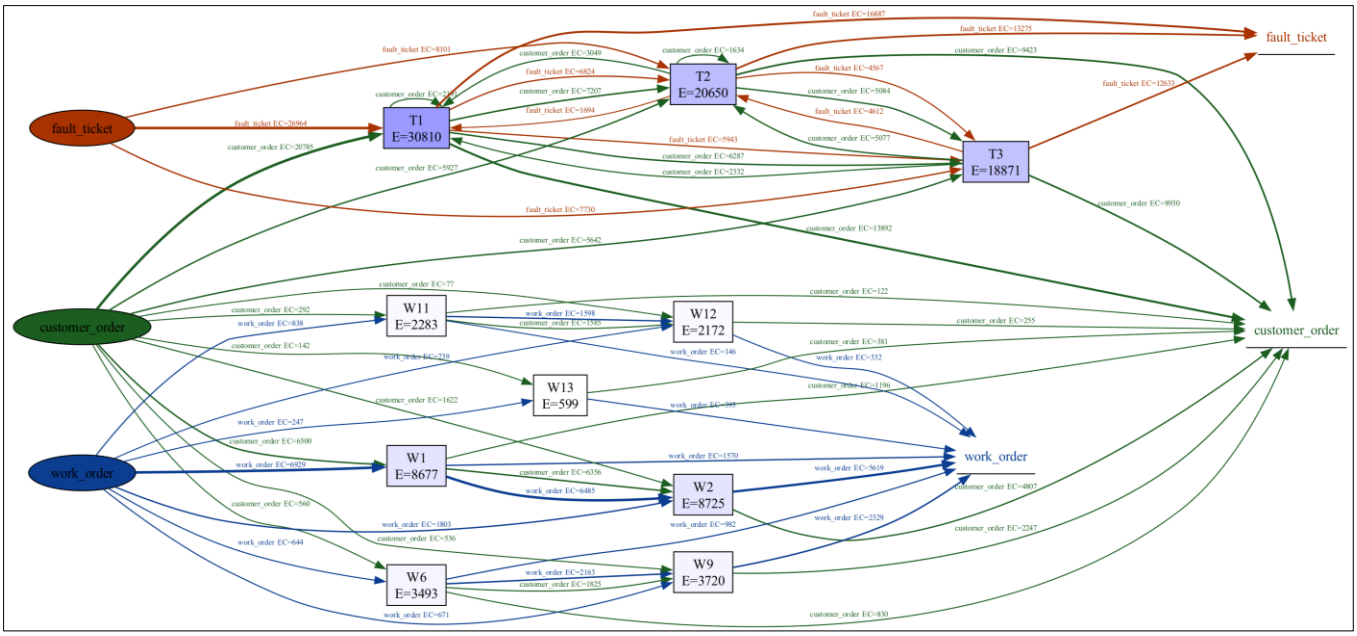


Figure 3. The discovered object-centric directly-follows graph (OC-DFG), drawn by pm4py

Note: Navy is the work-order lifecycle and rust is the fault-ticket lifecycle. Green is the customer order, whose edges span both. Restricted to the three object types that carry the argument and to the 19 directly-follows relations seen at least 1,500 times, out of 140 in those types. Activity labels are anonymised as described in Section 3.6

Table 7. Object-centricity profile by candidate case notion

Case Notion	Support	Deficiency	Conv. Factor	Conv. Rate	Divergence	Verdict
Work order	0.5000	0.5000	1.0000	0.0000	0.0000	deficient
Fault ticket	0.5000	0.5000	1.0000	0.0000	0.7539	divergent, deficient
Customer order	1.0000	0.0000	1.0000	0.0000	0.4375	divergent
Service	0.8202	0.1798	1.1648	0.1307	0.4784	convergent, divergent, deficient
Customer	0.9209	0.0791	1.0027	0.0027	0.5199	divergent

Three readings of Table 7 matter, and none of them is a restatement of a column. First, the conventional case notion is the most deficient one: flattening onto the work order, as every analysis of this process has done, including the authors' own, discards the entire fault lane because those events have no work order to attach to. Second, no candidate is safe. The one notion with full support pays for it with the highest divergence of any non-lifecycle type, so the choice is between losing events and merging objects. Third, the divergence metric earns its definition by separating repetition caused by collapsing distinct objects from repetition that is a genuine retry on one object: the work order shows heavy activity repetition and still scores zero, because W10 followed by W12 is one object retrying. A naive multiplicity count would have called both divergence and been wrong about the work order.

4.4 Model source and what each can see

Three models of the work order process were compared: the reference diagram transcribed by hand, its observable projection, and a model mined from the training split. A mined model's recall against the full diagram is bounded at 0.778 of nodes and 0.400 of edges by the two activities absent from the stream, and the measured values are exactly those bounds. The gap runs both ways: the log also records six activities the procedure does not draw, which together form the entire exception subprocess. The two sources are therefore complementary detectors. A mined model fits the log, failures included, so it cannot flag a failure; a normative model flags the exception traffic but states almost no durations, so it cannot detect a transition that is merely slow. The full comparison is

given in the Supplementary Material.

4.5 Cross-object detection

Table 8 reports one million events through the object-centric twin against the normative work order model and a mined ticket lifecycle.

Table 8. Deviations over 1,000,000 events, 3,988,900 relations, 1,600,517 objects

Deviation	Count
D1 unexpected transition	131,135
D2 skipped required	128,621
D7 unmodelled activity	79,372
off model	19,196
D9 parent state forbidden	13,580
D10 sibling stall	3,796
D4 loop exceeded	246

13,580 work orders reached a completing activity while a fault ticket raised against the same customer order was still open. Neither lane can produce that finding alone. The work order stream does not record tickets, and the ticket stream does not record work order status. It requires the relation recovered in Section 4.1 and a checker that can express a constraint between two lifecycles.

A further 3,796 cases had two tickets open concurrently against one customer order, which the model declares should not happen. Both counts need their denominators. The window holds 125,353 work orders and 381,917 fault tickets. Only 62,384 work orders, 49.8% of them, share a customer order with any fault ticket, and only those can raise D9 at all, so 13,580 firings are 21.8% of the eligible population rather than of all work orders. D10 is bounded by the 60,285 customer orders carrying more than one ticket, of which 3,796, or 6.3%, show the overlap. The remaining half of the work orders are not evidence of conformance: they are objects the relation does not reach in this window, and a rate measured on the linked subset need not hold on the unlinked one. Whether the unlinked objects behave differently cannot be established from this data, and Section 4.1 shows the shortfall is driven by a key the upstream system stopped writing rather than by anything about the work orders themselves. Mean per-object fitness is 0.4584 for work orders against the normative model and 1.0000 for tickets against their mined model, which is the expected asymmetry, since a mined model permits what occurred.

How many of these deviations are real? A count of disagreements with a model is not yet a count of problems, so a sample was adjudicated against evidence from outside the replay window. Six instances of each of the eight classes the twin raises in that window were drawn at random from a 200,000-event window opening at sequence 6,000,000, which is mid-stream and therefore able to contain objects whose earlier history falls outside it. Each of the 48 instances was checked against the object's complete history in the full log. Thirty-four, or 70.8%, are genuine: the object's whole history lies inside the window and still disagrees with the model. Thirteen are artefacts of the window opening part-way through a lifecycle, so the object only appears to start in the wrong state, and one is unresolved because the window closes before the object does. The artefacts are most frequent in skipped required and parent state forbidden, at three of six each, the two classes whose evidence depends most directly on an object's earlier history, and the single unresolved instance belongs to a class whose evidence is an absence; the per-class breakdown is given in the Supplementary Material. A window opening at the start of the log cannot exhibit this at all, which is why the check was not run on one. Two limits remain. The adjudication is rule-based rather than a domain expert's judgement, and it sees only the constructed log, so an object whose earlier events were never exported cannot be told apart from one that genuinely began where it appears to.

Two of the four interaction classes read zero here. D8 is zero because the customer order identifier parses from essentially every fault record, so no event lacks a required object; the linkage shortfall has a different shape and is reported as a reachability statistic rather than forced into the deviation channel, where at roughly two-thirds of events it would be noise. D11 is zero because the permitted cardinality for the service was set to the observed maximum, so by construction nothing exceeds it; setting that bound on the training split and measuring violations on the test split is the experimentally correct form and is left to future work. Both classes are exercised on fixtures, and both engines agree on them.

4.6 Engine throughput and the cost of object-centricity

Every figure in this subsection is the median of ten repetitions with the full range reported beside it, because a median over three runs cannot show whether a rate is stable. Table 9 gives single-case-notion throughput by rule count. Both engines consume identical prepared columns, and each repetition measures 500,000 events after a 50,000-event warm-up.

Table 9. Single-case-notion throughput in events per second, median of 10 repetitions with the observed range

Rules	Python Reference	Range	Rust	Range	Speed-up
1	847,573	3.7%	9,353,860	15.7%	11.0×
4	354,052	15.9%	7,861,856	24.4%	22.2×
8	136,156	12.9%	5,707,207	28.4%	41.9×
16	80,739	5.0%	5,043,207	13.7%	62.5×

The reference implementation degrades roughly linearly in rule count, while the compiled engine stays above five million events per second across the range, so the speed-up widens with rule count. The ranges are wide enough to matter: the compiled engine varies by up to 28% between repetitions, so ratios between engines should be read to one significant figure.

An independent contextual baseline places these figures against the state of practice, since both engines above were written by the same authors. pm4py's streaming token-based-replay checker, run on the same host over the same 100,000 events with its per-deviation logging disabled, sustains 17,935 events per second. This is not an apples-to-apples comparison, and the ratio is not reported as a speed-up: token-based replay searches for a firing sequence and accounts for missing and remaining tokens, whereas the compiled engine tests a precomputed state machine, which is the easier question. The figure is context for the order of magnitude involved, nothing more. No published online checker could serve as a baseline for the object-centric path, because none evaluates a constraint spanning two object types.

Table 10 decomposes the cost of object-centricity, because two things change at once: the degenerate configuration exercises the object-centric loop with exactly one object per

event, and the full configuration adds the real fan-out.

Table 10. The cost of object-centricity, 1,000,000 events at fan-out 3.99, median of 10 repetitions

Configuration	Events/s	ns per Event	ns per Relation	Range	vs Flat
Flat, single-case notion	10,303,135	97.1	97.1	13.0%	1.00×
Object-centric, one object per event	8,156,169	122.6	122.6	4.5%	1.26×
Object-centric, real fan-out	3,096,500	322.9	81.0	17.6%	3.33×

The per-relation cost of the object-centric path is 81.0 nanoseconds, below the flat path's 97.1 nanoseconds per event. The 3.33× slowdown per event is accounted for by there being 3.99 times as many objects to touch, at slightly lower cost per touch, so object-centricity on this workload costs what the additional object touches cost and nothing structural. These figures were measured on a one-million-event slice, where the longer measured region reduces the run-to-run range from roughly a third to under a fifth; the conclusion is unchanged at either scale. Cross-object checking stays cheap because the counts it needs are maintained as objects move: over one million events, the runtime performs 0.61 related-object visits per event. Per-event latency is also bounded, with a median of 83 and a 99th percentile of 834 nanoseconds for the compiled engine; the latency distribution and a throughput plot are given in the Supplementary Material.

4.7 Rule generation and two-view synthesis

This subsection reports a supporting component, not a contribution of the paper, and is kept deliberately short. Twenty-four natural-language requests, three paraphrases per underlying rule, including Indonesian variants, were put to the model with JSON-constrained decoding. The 24 requests cover eight underlying rules, 13 graded medium and 11 hard, spanning the patterns this operator asks for most: a state held too long, a transition slower than its envelope, a step out of order, a mandatory step skipped, a case cancelled after work began, repeated churn in one state, a case exceeding an absolute age, and a case already off-model. Eight rules over one stream are a preliminary probe rather than a benchmark, and the figures below characterise this prompt set on this vocabulary, not rule generation in general. All parse, satisfy the schema, and compile without a repair round, so the gate of Section 3.5 rejected nothing. Mean latency is 8.4 s. Exact structural match is 0.000, mean component agreement is 0.675, and mean behavioural agreement is 0.757, scored on the 21 of 24 gold rules that fire at least once in a one-million-event window.

Exact structural match being zero is not a failure, since behavioural agreement reaches 1.000 on several prompts whose rules are phrased differently from the gold rules. On a narrower 100,000-event window, 14 of 24 gold rules fire on nothing; widening to one million events makes the harder rules scorable. The narrower window therefore reported a higher mean over a smaller scorable subset.

The instructive failure is a dropped conjunct. The request “raise an anomaly when activation takes more than six hours after installation completed” has a gold rule with two conditions: the previous activity being W5 and the time in state exceeding six hours. The generated rule keeps the duration and drops the guard. It is valid JSON, satisfies the schema and compiles, and it fires 15,977 times against the gold rule's 94, matching every route into W6 that took longer than six hours instead of the one route asked about. The residual failures share this shape, and the omission is invisible to every stage that inspects form. Only simulation against history catches it, which is why the pipeline has that stage.

Two further observations are reported in full in the Supplementary Material. Running the model with its reasoning channel enabled made responses unreliable to parse, and in one run a request never returned, so the figures above have it disabled. The offline narrative that combines the flattened statistics with the object-centric results reproduced every number it was given, but misread two of them, which is why it is reported as a measured result and nothing in this paper rests on it.

The transferable result is negative. Syntactic validity does not imply semantic correctness: every generated rule is parsed, validated, and compiled, including the one that fired 15,977 times instead of 94. A language model can help an operator express a rule, but the draft must be replayed against history before it is armed, and it is the deterministic gate around the generator that makes the component safe to offer.

4.8 Threats to validity

Limitations argued in place above, including the replay setting, partial linkage and the contextual baseline, are not repeated here; this subsection states only what qualifies the results as a whole.

One operator, one process. Ten million events and 38.8 million relations are a large sample of one order fulfilment process at one company, not a sample of the industry. The structural findings should travel where the same shape holds: a commercial order spawning several fulfilment objects that separate systems track on separate clocks, as is common in telecommunications, utilities and logistics. The specific quantities should not: a deficiency of 0.50 follows from this operator's two lanes being of almost equal size, and the coverage shortfall from one upstream system's behaviour. Replication on a second operator or a public object-centric log is the obvious next test and has not been done here.

Both engines are the authors' own. The gate proves they compute the same function, but it remains a comparison of two implementations by the same authors, which is why the contextual pm4py baseline is reported beside them.

Ground truth is the weakest link. No labelled anomaly set exists for this data, so what is evaluated is agreement with a documented specification and with recorded outcomes. The adjudication in Section 4.5 is rule-based rather than a domain expert's judgement, and the gold rules and generator prompt in Section 4.7 were written by the same author, which is why the behavioural metric is the one to trust there.

5. CONCLUSIONS

Order fulfilment at this operator is a discrete event system with several interacting object lifecycles, and treating it as one is measurably lossy: flattening onto the work order, the case notion every prior analysis of this process has used, discards

half of all events, and no candidate case notion is free of all three flattening pathologies. The relation that makes the object-centric view possible, reported unavailable by an earlier internal analysis, links 539,248 pairs and 2.94 million events once routed through the customer order, and linked work orders fail substantially more often than unlinked ones.

A constraint spanning two lifecycles proved both expressible and cheap: in one million events, 13,580 work orders reached a completing activity while a ticket against the same order was still open, at 0.61 related-object visits per event. Object-centricity is also affordable: at 81.0 nanoseconds per event-to-object relation against 97.1 nanoseconds per event for the single-case-notion path, both medians of ten repetitions, the additional cost is the additional objects, not the machinery for handling them.

What this paper demonstrates and what it does not should be kept apart. Demonstrated: an object-centric log built from industrial exports with its coverage measured; the cost of flattening quantified for every candidate case notion; four inter-type deviation classes specified, implemented and checked identically by two independent engines; and per-event cost measured under controlled replay. Not demonstrated: operation against a live feed, since every figure comes from replaying historical logs and no production work order was touched; complete linkage, since the relation reaches 34.0% of work orders and the cross-object rates are properties of that subset; and validated anomaly detection, since agreement with a documented specification is not the same as detecting anomalies an expert would confirm, and the adjudication in Section 4.5 is rule-based rather than expert judgement.

Three directions follow. An uncapped extract would raise the coverage on which the cross-object findings rest. The permitted cardinality of a convergent object type should be set on a training split, with violations measured on a held-out one, turning a bound that currently cannot be exceeded into a measurement. Finally, a rule set authored by an independent domain expert would remove the semi-circularity of the present generation measurement.

ACKNOWLEDGMENT

The authors thank the state-owned telecommunications company that gave permission to work on their fault management data. At the company's request, the operator is not named, and every activity label is a published code, as set out in Section 3.6; the mapping to the operator's own labels is not published. This research was funded by Telkom University.

REFERENCES

[1] Chen, C.M. (2016). Use cases and challenges in telecom big data analytics. APSIPA Transactions on Signal and Information Processing, 5: e19. https://doi.org/10.1017/ATSIP.2016.20

[2] van der Aalst, W.M.P., Adriansyah, A., Alves de Medeiros, A.K., et al. (2012). Process mining manifesto. In International conference on business process management, Clermont-Ferrand, France, pp. 169-194. https://doi.org/10.1007/978-3-642-28108-2_19

[3] van der Aalst, W.M.P. (2016). Process Mining: Data Science in Action, 2nd ed. Springer, Berlin. https://doi.org/10.1007/978-3-662-49851-4

[4] Kusuma, G.P., Kurniati, A.P., Rojas, E., McInerney, C.D., Gale, C.P., Johnson, O.A. (2021). Process mining of disease trajectories: A literature review. Studies in Health Technology and Informatics, 281: 457-461. https://doi.org/10.3233/SHTI210200

[5] Kurniati, A.P., Wisudiawan, G.A.A., Kusuma, G.P., et al. (2024). Patient clustering to improve process mining for disease trajectory analysis using Indonesia health insurance dataset. In Proceedings of the 7th International Conference on Artificial Intelligence and Big Data, Chengdu, China, pp. 88-93. https://doi.org/10.1109/ICAIBD62003.2024.10604436

[6] van der Aalst, W.M.P. (2019). Object-centric process mining: Dealing with divergence and convergence in event data. In International Conference on Software Engineering and Formal Methods, Oslo, Norway, pp. 3-25. https://doi.org/10.1007/978-3-030-30446-1_1

[7] Ghahfarokhi, A.F., Park, G., Berti, A., van der Aalst, W.M.P. (2021). OCEL: A standard for object-centric event logs. In European Conference on Advances in Databases and Information Systems, Tartu, Estonia, pp. 169-175. https://doi.org/10.1007/978-3-030-85082-1_16

[8] van der Aalst, W.M.P., Berti, A. (2020). Discovering object-centric Petri nets. Fundamenta Informaticae, 175(1-4): 1-40. https://doi.org/10.3233/FI-2020-1946

[9] Adams, J.N., van der Aalst, W.M.P. (2021). Precision and fitness in object-centric process mining. In 2021 3rd International Conference on Process Mining, IEEE, pp. 128-135. https://doi.org/10.1109/ICPM53251.2021.9576886

[10] Berti, A., Park, G., Rafiei, M., van der Aalst, W.M.P. (2023). A generic approach to extract object-centric event data from databases supporting SAP ERP. Journal of Intelligent Information Systems, 61(3): 835-857. https://doi.org/10.1007/s10844-023-00799-9

[11] Egger, A., Fehrer, T., Kratsch, W., Wördehoff, N., König, F., Röglinger, M. (2025). Refining the process picture: Unstructured data in object-centric process mining. Information Systems, 134: 102582. https://doi.org/10.1016/j.is.2025.102582

[12] Xiu, B., Li, G. (2023). Diagnosing conformance between object-centric event logs and models. IEEE Access, 11: 110837-110849. https://doi.org/10.1109/ACCESS.2023.3322366

[13] Park, G., van der Aalst, W.M.P. (2025). Operational process monitoring: An object-centric approach. Computers in Industry, 164: 104170. https://doi.org/10.1016/j.compind.2024.104170

[14] Liss, L., Adams, J.N., van der Aalst, W.M.P. (2023). Object-centric alignments. In Conceptual Modeling, International Conference on Conceptual Modeling, Lisbon, Portugal, pp. 201-219. https://doi.org/10.1007/978-3-031-47262-6_11

[15] Liss, L., van der Aalst, W.M.P. (2026). Object-centric conformance checking on object-centric causal nets. In Advanced Information Systems Engineering, Lecture Notes in Computer Science, Verona, Italy, pp. 331-347. https://doi.org/10.1007/978-3-032-28117-3_19

[16] Berti, A., van der Aalst, W.M.P. (2023). OC-PM: Analyzing object-centric event logs and process models. International Journal on Software Tools for Technology Transfer, 25(1): 1-17. https://doi.org/10.1007/s10009-022-00668-w

[17] Park, G., Adams, J.N., van der Aalst, W.M.P. (2022). OPerA: Object-centric performance analysis. In Conceptual Modeling, Lecture Notes in Computer Science, Hyderabad, India, pp. 281-292. https://doi.org/10.1007/978-3-031-17995-2_20

[18] Adriansyah, A., van Dongen, B.F., van der Aalst, W.M.P. (2011). Conformance checking using cost-based fitness analysis. In Proceedings of the 15th IEEE International Enterprise Distributed Object Computing Conference, Helsinki, Finland, pp. 55-64. https://doi.org/10.1109/EDOC.2011.12

[19] Burattin, A., van Zelst, S.J., Armas-Cervantes, A., van Dongen, B.F., Carmona, J. (2018). Online conformance checking using behavioural patterns. In 16th International Conference on Business Process Management, Sydney, NSW, Australia, pp. 250-267. https://doi.org/10.1007/978-3-319-98648-7_15

[20] van Zelst, S.J., Bolt, A., Hassani, M., van Dongen, B.F., van der Aalst, W.M.P. (2019). Online conformance checking: Relating event streams to process models using prefix-alignments. International Journal of Data Science and Analytics, 8(3): 269-284. https://doi.org/10.1007/s41060-017-0078-6

[21] Awad, A. (2026). Conformance checking on event streams: A comprehensive survey. Computer Science Review, 63: 101051. https://doi.org/10.1016/j.cosrev.2026.101051

[22] Schuster, D., van Zelst, S.J. (2020). Online process monitoring using incremental state-space expansion: An exact algorithm. In 18th International Conference on Business Process Management, Seville, Spain, pp. 147-164. https://doi.org/10.1007/978-3-030-58666-9_9

[23] Lee, W.L.J., Burattin, A., Munoz-Gama, J., Sepúlveda, M. (2021). Orientation and conformance: A HMM-based approach to online conformance checking. Information Systems, 102: 101674. https://doi.org/10.1016/j.is.2020.101674

[24] Zaman, R., Hassani, M., van Dongen, B.F. (2023). Conformance checking of process event streams with constraints on data retention. Information Systems, 117: 102228. https://doi.org/10.1016/j.is.2023.102228

[25] Krajsic, P., Franczyk, B. (2021). Variational autoencoder for anomaly detection in event data in online process mining. In Proceedings of the 23rd International Conference on Enterprise Information Systems, pp. 567-574. https://doi.org/10.5220/0010375905670574

[26] Effendi, Y.A., Kim, M. (2025). Reliable process tracking under incomplete event logs using timed genetic-inductive process mining. Systems, 13(4): 229. https://doi.org/10.3390/systems13040229

[27] Dumas, M., Fournier, F., Limonad, L., et al. (2023). AI-augmented business process management systems: A research manifesto. ACM Transactions on Management Information Systems, 14(1): 11. https://doi.org/10.1145/3576047

[28] Martino, D., Perlangeli, C., Grottoli, B., La Rosa, L., Pacella, M. (2025). A knowledge-driven framework for AI-augmented business process management systems: Bridging explainability and agile knowledge sharing. AI, 6(6): 110. https://doi.org/10.3390/ai6060110

[29] Chapela-Campa, D., Dumas, M. (2023). From process mining to augmented process execution. Software and Systems Modeling, 22(6): 1977-1986. https://doi.org/10.1007/s10270-023-01132-2

[30] Ceravolo, P., Comuzzi, M., De Weerdt, J., Di Francescomarino, C., Maggi, F.M. (2024). Predictive process monitoring: Concepts, challenges, and future research directions. Process Science, 1(1): 2. https://doi.org/10.1007/s44311-024-00002-4

[31] Galanti, R., de Leoni, M., Monaro, M., et al. (2023). An explainable decision support system for predictive process analytics. Engineering Applications of Artificial Intelligence, 120: 105904. https://doi.org/10.1016/j.engappai.2023.105904

[32] Rizzi, W., Comuzzi, M., Di Francescomarino, C., et al. (2024). Explainable predictive process monitoring: A user evaluation. Process Science, 1(1): 3. https://doi.org/10.1007/s44311-024-00003-3

[33] Fahrenkrog-Petersen, S.A., Tax, N., Teinemaa, I., et al. (2022). Fire now, fire later: Alarm-based systems for prescriptive process monitoring. Knowledge and Information Systems, 64(2): 559-587. https://doi.org/10.1007/s10115-021-01633-w

[34] Shoush, M., Dumas, M. (2025). White box specification of intervention policies for prescriptive process monitoring. Data & Knowledge Engineering, 155: 102379. https://doi.org/10.1016/j.datak.2024.102379

[35] Fioretto, S., Masciari, E. (2025). A comparative analysis of predictive process monitoring: Object-centric versus classical event logs. Knowledge and Information Systems, 67(9): 7355-7398. https://doi.org/10.1007/s10115-025-02461-y

[36] Cugola, G., Margara, A. (2012). Processing flows of information: From data stream to complex event processing. ACM Computing Surveys, 44(3): 15. https://doi.org/10.1145/2187671.2187677

[37] Forgy, C.L. (1982). Rete: A fast algorithm for the many pattern/many object pattern match problem. Artificial Intelligence, 19(1): 17-37. https://doi.org/10.1016/0004-3702(82)90020-0

[38] Grieves, M., Vickers, J. (2017). Digital twin: Mitigating unpredictable, undesirable emergent behavior in complex systems. In Transdisciplinary Perspectives on Complex Systems, Springer, pp. 85-113. https://doi.org/10.1007/978-3-319-38756-7_4

[39] Tao, F., Zhang, H., Liu, A., Nee, A.Y.C. (2019). Digital twin in industry: State-of-the-art. IEEE Transactions on Industrial Informatics, 15(4): 2405-2415. https://doi.org/10.1109/TII.2018.2873186

[40] Aheleroff, S., Xu, X., Zhong, R.Y., Lu, Y. (2021). Digital twin as a service (DTaaS) in Industry 4.0: An architecture reference model. Advanced Engineering Informatics, 47: 101225. https://doi.org/10.1016/j.aei.2020.101225

[41] Park, G., van der Aalst, W.M.P. (2021). Realizing a digital twin of an organization using action-oriented process mining. In 2021 3rd International Conference on Process Mining, Eindhoven, Netherlands, pp. 104-111. https://doi.org/10.1109/ICPM53251.2021.9576846

[42] Karakra, A., Fontanili, F., Lamine, E., Lamothe, J. (2022). A discrete event simulation-based methodology for building a digital twin of patient pathways in the hospital for near real-time monitoring and predictive simulation. Digital Twin, 2: 1. https://doi.org/10.12688/digitaltwin.17454.1

[43] Kremer, G., Sikut, E.M., Ribeiro, L.R.B., Dallavalle, S., Stark, R. (2026). Digital process twin classification and software selection. In International Conference on Advanced Information Systems Engineering, Verona,

Italy, pp. 67-78. https://doi.org/10.1007/978-3-032-28160-9_7

[44] Chen, M., Tworek, J., Jun, H., et al. (2021). Evaluating large language models trained on code. arXiv preprint arXiv:2107.03374. https://doi.org/10.48550/arXiv.2107.03374

[45] Rajkumar, N., Li, R., Bahdanau, D. (2022). Evaluating the text-to-SQL capabilities of large language models. arXiv preprint arXiv:2204.00498. https://doi.org/10.48550/arXiv.2204.00498

[46] Rebmann, A., Schmidt, F.D., Glavaš, G., van der Aa, H. (2025). On the potential of large language models to solve semantics-aware process mining tasks. Process Science, 2(1): 10. https://doi.org/10.1007/s44311-025-00019-3

[47] Casciani, A., Bernardi, M.L., Cimitile, M., Marrella, A. (2026). Enabling natural language analysis for object-centric event logs. Process Science, 3: 5. https://doi.org/10.1007/s44311-026-00037-9

[48] Casciani, A., Bernardi, M.L., Cimitile, M., Marrella, A. (2024). Conversational systems for AI-augmented business process management. In International Conference on Research Challenges in Information Science, Guimarães, Portugal, pp. 183-200. https://doi.org/10.1007/978-3-031-59465-6_12

NOMENCLATURE

AI	artificial intelligence
BPMN	Business Process Model and Notation
D1–D11	deviation classes raised by the online conformance checker (Tables 1 and 2)
JSON	JavaScript Object Notation
OC-DFG	object-centric directly-follows graph
OCEL	object-centric event log
OCPN	object-centric Petri net
T1–T3	published labels for the fault-ticket lifecycle: open, resolved, closed
W1–W15	published labels for the work-order lifecycle activities; W1 is the initial approval state, W9 normal completion, W10 the failure state, and W3 and W4 the two steps this stream cannot record
support(t)	share of events related to at least one object of type t
deficiency(t)	1 − support(t), share of events flattening onto type t discards
conv. factor(t)	flattened rows per covered event, the duplication a flattening incurs
conv. rate(t)	share of events relating to more than one object of type t
divergence(t)	share of repeated activities explained by collapsing distinct objects rather than by a loop
fan-out	mean number of objects an event relates to
ρ_skip	|D5| / (|D1| + |D5|), share of apparent non-conformance that is unobservability