Research Article

Making Operational States Explicit: A Unified Framework for State-Aware Object-Centric Process Mining

Alessandro Berti[1], Dina Kretzschmann[1,*], and Wil M. P. van der Aalst[1,2]

[1]Process and Data Science (PADS), RWTH Aachen University, Aachen, Germany
[2]Celonis, Munich, Germany

[*]Corresponding author: Dina Kretzschmann. Email: dina.kretzschmann@pads.rwth-aachen.de.

Abstract
The same activity can have different operational meanings depending on an object's condition. Object-centric process mining records events and their participating objects, but conditions such as understock or machine degradation usually remain implicit in attributes and relations. We present State-Aware Object-Centric Process Mining (SA-OCPM), a framework that makes these conditions explicit through a versioned state notion for a selected object type. The formalization connects event-object annotations and state calendars to transitions, episodes, state-aware directly-follows graphs, and recurring behavioral patterns. Auditable domain expressions and learned execution regimes share the same state interface. FLOWVAULT, a browser-based workbench for Object-Centric Event Log (OCEL) 2.0 data, implements the main analyses. Two synthetic experiments, covering inventory management and manufacturing with predictive maintenance, evaluate the implementation against independently computed references. Expression-based assignments and transitions match these references exactly and expose dwell, recovery, recurrence, and state-conditioned behavior. Exploratory prediction comparisons favor state features on several tasks, although raw features remain competitive and precomputed learned representations limit conclusions about prospective performance. Automatically discovered states have high within-cell purity but weak overall agreement with simulator regimes, limited label transfer, and substantial false-alarm burden in manufacturing. Pattern retrieval and broad conformance rules reveal further limitations. Tested native-core operations complete within seconds on logs of up to 300,000 events. The results support expression-based state analysis under explicit temporal and data-quality assumptions, while learned states require further validation before operational use.

Keywords: object-centric process mining; state-aware process mining; OCEL 2.0; behavioral patterns; self-organizing maps; process-aware decision support

1 Introduction

Background. Process mining turns the event data recorded by information systems into evidence about how processes are actually executed. Classical techniques assume that every event belongs to exactly one case, such as an order or a patient. Real processes rarely respect this assumption. A single goods receipt may concern a purchase order, several order items, a delivery, material items, storage locations, and a supplier; a single maintenance event may concern a machine, a work order, a production order, a component, and an operator. *Object-centric process mining* (OCPM) resolves this mismatch by keeping events connected to all participating objects, so that the behavior of different object types can be analyzed without flattening the data into one artificial case notion [Berti et al., 2024b, Ghahfarokhi et al., 2021, van der Aalst, 2019, 2023].

Problem. Object-centric event logs (OCELs) provide the data foundation for such analyses. OCEL 2.0 records event attributes, time-varying object attributes, event-to-object relations, and object-to-object relations. Yet this expressiveness leaves open the question that dominates operational practice: in which condition was the object when the event happened? We call this condition the *operational state* of an object, for example `Understock`, `Normal`, or `Overstock` for an inventory item, or `Running`, `Degraded`, `Down`, or `Recovery` for a machine. The evidence for the state is usually present in the log, but it is scattered over several fields, hidden behind thresholds, valid only during time intervals, or never recorded as one explicit label. As a consequence, the state changes that decision makers care about most remain *latent* in exactly the data that was collected to reveal them.

The missing state abstraction limits which questions can be answered. Without SA-OCPM, or an equivalent explicit reconstruction of operational states, a state-agnostic analysis cannot measure time in `Understock`, distinguish sustained recovery from a temporary stock increase, or group behavior by entry into and exit from `Degraded`. These results require a declared state definition, an object to which it applies, and temporal

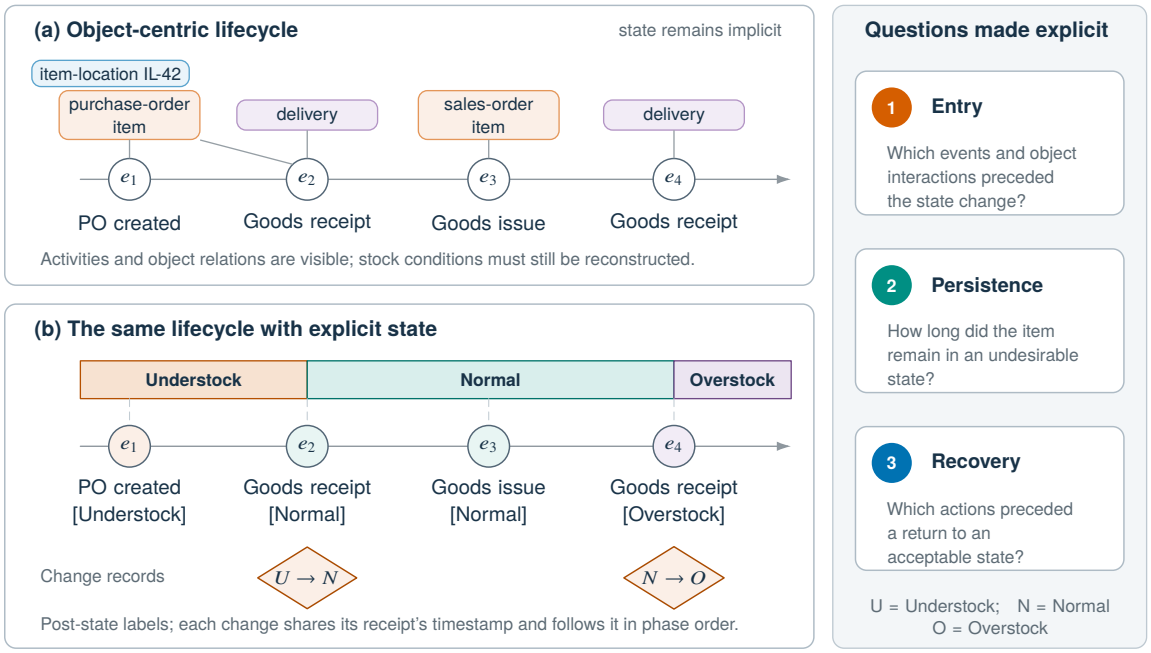

Fig. 1 Motivating item-location example. The same object-centric evidence is shown first without an explicit operational state and then with a state calendar, state-change records, and post-state activity labels. State awareness makes entry, persistence, and recovery questions explicit while preserving the base events and object relations.

boundaries. The raw OCEL can contain all the necessary evidence, but an activity-based variant or ordinary OCDFG does not supply these semantics. SA-OCPM makes them explicit and reusable across analyses.

Motivating example. The cost of leaving these conditions implicit is easy to underestimate. Consider an item at one storage location. The same activity, Goods Receipt, is beneficial when the item is understocked, routine when stock is normal, and harmful when stock is already excessive. A sequence of Purchase Order Created, Goods Receipt, and Goods Issue may therefore describe a recovery, a stable replenishment cycle, or an overstock escalation, and a traditional directly-follows graph cannot tell these situations apart because the activity names are identical. What analysts actually need to know is when the item entered an undesirable state, how long it remained there, which object interactions surrounded the transition, whether the observed behavior conforms to the policy for that state, and which local patterns repeatedly maintain or resolve the condition. Figure 1 illustrates this contrast on the example used throughout the paper. The upper panel shows an ordinary object-centric lifecycle in which events and related objects are visible but the operational condition remains invisible. The lower panel overlays a *state calendar*, explicit state changes, and state-conditioned activity labels on the same base evidence, and the diagnostic cards show how entry, persistence, and recovery questions become directly answerable. State thus acts as an *analytical coordinate system* over behavior, not as a display decoration.

Two routes to states. The same problem occurs in manufacturing, where an inspection during normal operation differs from an inspection during a degraded regime, and a maintenance action during planned setup differs from the same action during an unplanned stoppage. These examples reveal that states can originate in two ways. When accepted domain semantics exist, such as engineering rules over alarms, vibration thresholds, and machine modes, an *expression-based* route can turn them into auditable rules. When no accepted taxonomy exists, an *automatically discovered* route can extract recurrent execution regimes and their boundary conditions from lifecycle windows of activities, measurements, and object interactions. In practice both routes meet in a *hybrid* workflow in which analysts inspect, name, merge, split, or refine discovered states. Any credible framework for state-aware analysis must support all three.

Gap. State-Aware Object-Centric Process Mining (SA-OCPM) was introduced to make state evolution explicit in object-centric event data by adding state transition events and state-aware activity context [Kretzschmann et al., 2026b]. Subsequent work contributed segmentation of long object lifecycles [Berti et al., 2026b], automatic execution-state abstraction using principal component analysis and self-organizing maps [Berti et al., 2026c], structural inventory analysis using time-in-state outcomes [Berti et al., 2026a], and automated detection of

state-aware behavioral patterns [Kretzschmann et al., 2026a]. Each contribution establishes one building block, but the blocks rest on different representations and were designed for specific analytical goals. What is missing is a single formal foundation with an explicit treatment of ambiguity, a structured catalog of state-aware analyses, and a transparent mapping from concepts to tool support and empirical evaluation. This paper provides that unified account.

Research questions. The paper is guided by four research questions:

RQ1. How can operational states be represented as a *conservative extension* of object-centric event data and object-centric directly-follows graphs, so that no base evidence is lost?

RQ2. How can state notions be defined manually or detected automatically while retaining *provenance* and interpretability?

RQ3. Which analyses become possible when *state persistence* and *state change* are first-class objects of study?

RQ4. How can the framework be operationalized in a tool and evaluated across domains, and where are its current limits?

Contributions. In answering these questions, the paper makes five contributions:

1. **A unified formalization.** Starting from an OCEL and an object-centric directly-follows graph (OCDFG), we introduce a leading-object-specific *state notion* and derive a state-aware OCEL (SA-OCEL), a state-aware OCDFG (SA-OCDFG), *state episodes*, *state-determined segments*, and recurring behavioral patterns. The formalization uses *event-object incidences* as the primary carrier of state, which avoids the hidden assumption that one event shared by several leading objects must have the same state for all of them.

2. **Interchangeable state construction.** Expression-based state notions evaluate ordered conditions over event attributes, time-valid object attributes, object relations, and derived quantities. Automatic state notions encode lifecycle windows, reduce the representation, and map the windows to execution states. Both routes produce the same abstract state interface, so every downstream analysis is independent of how states were obtained.

3. **A structured analysis catalog.** We organize state-aware analyses by the question being answered: state occupancy, transition frequencies, dwell and recovery times, state-conditioned control flow, conformance against state policies, intra-state and inter-state pattern detection, boundary-condition analysis, association and rule mining, structural and causal modeling under explicit assumptions, prediction of undesirable transitions, and evaluation of candidate interventions. For each family, we state what the state perspective adds over a state-agnostic analysis.

4. **Tool support.** The FLOWVAULT workbench implements the framework in the browser. It combines an Angular interface with a Rust and WebAssembly core, imports and exports several OCEL 2.0 formats, supports both state-construction routes, and exposes state-aware graphs, transition indicators, time perspectives, patterns, lifecycle details, feature associations, and an exploratory causal workbench.

5. **A reproducible, oracle-backed evaluation.** Two synthetic experiments, one on inventory management and one on manufacturing with predictive maintenance, assess state and transition validity, operational and predictive utility, automatic-state quality and transfer, pattern and conformance behavior, robustness to plausible perturbations, causal-pipeline sanity checks, and computational performance. The assessment deliberately reports negative results and identifies which human-centered questions remain open.

Scope. The scope of the paper is deliberately analytical. A state is a discrete abstraction that is useful for a specified objective and leading object type. It is not assumed to be the unique true physical state of an object, and a learned state is not automatically a causal mechanism. Different objectives can justify different partitions of the same lifecycle: a replenishment analyst may distinguish `Understock`, `Normal`, and `Overstock`, while a finance analyst may distinguish `Low Value`, `High Value`, and `Blocked` for the same order population. The state notion and its *provenance* must therefore always be made explicit.

Outline. The remainder of the paper is structured as follows. Section 2 positions the work relative to object-centric, data-aware, state-based, local-pattern, and diagnostic process mining. Section 3 defines OCELs, object lifecycles, and OCDFGs without states. Section 4 presents the SA-OCPM framework from state construction to analysis. Section 5 describes FLOWVAULT. Section 6 defines the evaluation questions and the two experimental scenarios. Section 7 reports the assessment, Section 8 discusses its implications and limitations, and Section 9 concludes the paper.

Applications beyond the evaluation scenarios. The same construction applies wherever an operational condition can be assigned to an identifiable object over time. Figure 2 illustrates five prospective settings beyond inventory management and manufacturing. Each setting pairs a possible leading object with a state vocabulary, related objects, and a question about persistence or change. These are illustrative applications of the framework, not additional empirical studies; state definitions would need to be agreed and validated in each setting. Practical guidance for choosing the leading object type is given in Section 4.1.1.

2 Related Work

2.1 Object-centric event data and process models

The motivation for OCPM is the divergence and convergence problem that arises when event data involving multiple entities are forced into one case identifier [van der Aalst, 2019]. Flattening around purchase orders may duplicate events shared by multiple items, while flattening around items may lose relations to deliveries or suppliers. OCPM instead keeps events and objects in one connected data structure and derives perspectives when needed. This change affects discovery, conformance, performance, filtering, feature extraction, and variant analysis.

The OCEL standard established a common representation for object-centric event data [Ghahfarokhi et al., 2021]. OCEL 2.0 extends the representation with time-varying object attributes, qualified event-to-object and object-to-object relations, and several interoperable storage formats [Berti et al., 2024b, Koren et al., 2024]. These features are directly relevant to states. For example, an object's stock level, status, risk class, or machine mode can change over time, and a state expression must resolve the value that is valid at the point being analyzed. Nevertheless, the standard is intentionally neutral about which attributes define an operational state and how a state change should be represented analytically.

Several object-centric process models have been proposed. Object-centric Petri nets provide formal behavioral models and express synchronization across object types [van der Aalst and Berti, 2020]. OCDFGs provide a

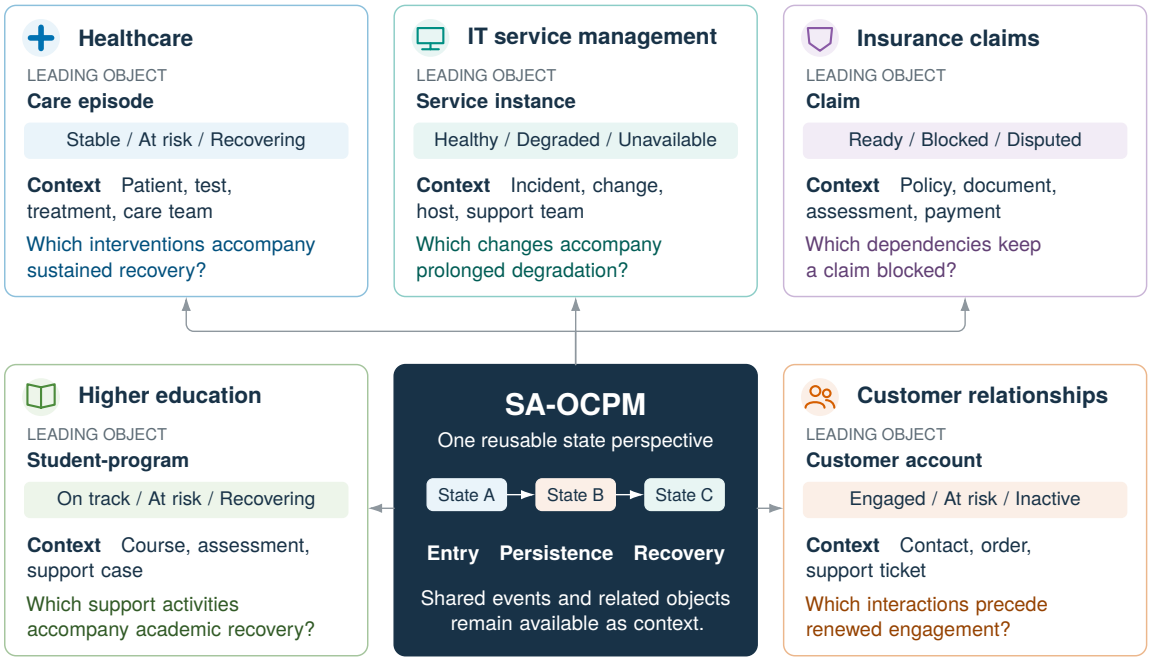

Fig. 2 Illustrative SA-OCPM applications beyond the two evaluation scenarios. Each card gives a candidate leading object, operational states, object context, and a state-aware question. The central abstraction is shared; the domain-specific state definitions and their validity remain modeling choices.

simpler graph abstraction in which directly-follows edges are typed by the object lifecycle that induces them. They are useful for exploration and have been extended to conformance and performance analysis [Berti and van der Aalst, 2023, Park et al., 2024]. The present work builds first on OCDFGs because their local edge semantics make the effect of state conditioning transparent, while the state annotations developed here can also be used with richer object-centric models.

Object-centric feature extraction and tool support help transform interconnected event data into analysis-ready representations. OC-PM provides exploration, filtering, process discovery, and conformance functions for OCELs [Berti and van der Aalst, 2023]. Graph-based feature extraction summarizes object behavior and interactions for machine-learning tasks [Berti et al., 2024a]. These methods are complementary to SA-OCPM. Features can be inputs to automatic state discovery, and state exposure or state-transition measures can become outputs or additional features.

2.2 Data-aware, multi-perspective, and state-based process mining

Classical data-aware process mining relates control-flow behavior to data conditions. Decision discovery identifies attributes that explain routing decisions, often using alignments and classifiers [de Leoni and van der Aalst, 2013]. Declarative approaches express constraints that involve both events and data values [Maggi et al., 2023]. Object-centric data-awareness extends similar concerns to data that belong to several interacting objects and that evolve independently [Goossens et al., 2023]. These contributions show that behavior cannot be interpreted from activity order alone.

SA-OCPM differs from treating every attribute value as an additional event dimension. A state notion deliberately compresses potentially many observations into a finite operational regime with temporal continuity. This allows persistence, entry, exit, recurrence, and recovery to be studied directly. The compression also supports a stable vocabulary for decision making, such as `Understock` or `Machine Degraded`, even when the underlying evidence consists of several fields and relations.

State-based models for multi-perspective processes have previously been discovered from conventional event logs [van Eck et al., 2016]. Lifecycle-based process performance analysis also emphasizes the phases through which a case passes and the time spent in them [Hompes and van der Aalst, 2018]. Dynamic process models and time-series abstractions similarly connect changing measurements with process behavior. These works demonstrate the value of state and phase abstractions, but they typically start from one case or one principal entity. SA-OCPM must additionally preserve events that refer to multiple object types and must define which object's state is being used to interpret a shared event.

A second distinction concerns the source of the state. In some settings, state labels follow from accepted rules. In others, the state space itself is unknown. Principal component analysis (PCA) provides a linear low-dimensional representation of correlated features [Jolliffe and Cadima, 2016], while self-organizing maps (SOMs) provide a topology-preserving discretization that can be inspected as a two-dimensional grid [Kohonen, 1990]. The execution-state abstraction of Berti et al. [2026c] combines lifecycle windows, PCA, and a SOM to discover object-centric regimes and to characterize the windows at state boundaries. The current paper embeds that method into a more general definition of a state notion and makes it interchangeable with manual expressions.

2.3 Object-centric executions, variants, segments, and local patterns

Global process models are not always the best unit for diagnosis. Object-centric executions and variants define comparison units without reducing the log to one conventional case [Adams et al., 2022]. Super variants aggregate related variants to support broader summaries of behavior [Adams et al., 2024]. Temporal object type models and advanced case notions provide other ways to organize interconnected execution data. These approaches primarily group complete or selected executions by control-flow structure.

Long-lived objects create a related challenge. A material-location object or machine may accumulate thousands of events over months or years, while a problem is confined to a short episode. Segmentation approaches split such lifecycles according to leading-child relations, event types, or temporal intervals and then create segment-level feature tables [Berti et al., 2026b]. SA-OCPM adds a domain-relevant segmentation boundary: a change in the selected state notion. State-determined segments therefore have an explicit analytical interpretation, such as behavior inside `Understock` or behavior around a `Normal` to `Overstock` transition.

Behavioral-pattern mining focuses on recurring local behavior rather than complete executions. Object-centric local process models represent recurring behavior involving several object types with object-centric Petri nets [Peeva et al., 2025]. Drill-down and roll-up methods expose patterns at different dimensions and levels of granularity [Miri and Jalali, 2025]. Rule-mining approaches connect object-centric features and behavior with outcomes for root-cause analysis [Knopp et al., 2025]. These techniques are valuable, but the selected local behavior is not necessarily aligned with state persistence or state change.

The state-aware pattern approach of Kretzschmann et al. [2026a] begins with the episodes induced by a leading object's state sequence. It constructs local graphs for episodes within one state and for adjacent episodes spanning a change, groups structurally equivalent graphs, and ranks the resulting patterns by support and control-flow mass. The method therefore distinguishes behavior that maintains a condition from behavior that accompanies entry into or recovery from it. Section 4.7 formalizes this relationship and shows how it fits the broader SA-OCPM framework.

2.4 Diagnostic, causal, and decision-support perspectives

Object-centric conformance and performance analysis can reveal deviations and delays without discarding the participating object types [Park et al., 2024]. Root-cause approaches use rules, correlations, or learned models to connect process features with outcomes [Knopp et al., 2025]. Structural equation models have also been used in process mining to represent hypothesized relations among operational factors [Qafari and van der Aalst, 2020]. These methods move process mining from description toward explanation, but their results depend strongly on how the outcome is defined.

State exposure provides a natural class of outcomes. In inventory management, the fraction of observed time spent in `Understock` and `Overstock` is more informative than a binary label stating that either state occurred at least once. Berti et al. [2026a] reconstruct inventory trajectories, derive indicators for demand, supply, batchiness, and buffering, and relate them to `Understock` and `Overstock` exposure using a theory-constrained structural equation model. This illustrates how SA-OCPM can feed a decision-support analysis. It does not imply that the event log alone identifies causal effects. Causal interpretation still requires a justified graph, adequate measurement, control of confounding, temporal precedence, and sensitivity analysis.

Decision support can also be based on state-conditioned policies. A discovered execution state can be linked to a stock regime, characteristic boundary conditions, and candidate actions [Berti et al., 2026c]. A rule-based state can be checked against allowed actions, escalation paths, or service-level targets. The advantage is that recommendations are attached to an operational context that users recognize. The risk is that a state label may hide uncertainty or combine several mechanisms. The framework therefore stores the provenance of the state notion and treats confidence, stability, and interpretability as evaluation criteria.

2.5 Positioning of this paper

The original SA-OCPM paper introduced explicit state transitions and state-aware events [Kretzschmann et al., 2026b]. The automatic abstraction paper proposed a way to discover states and boundary conditions [Berti et al., 2026c]. The patterns paper proposed state-determined local graph aggregation [Kretzschmann et al., 2026a]. The segmentation and causal inventory papers contributed complementary analysis units and outcomes [Berti et al., 2026a,b]. The present paper consolidates these elements and extends them in four ways: it provides a single end-to-end formalization; it uses event-object incidences to handle events related to several leading objects; it separates the abstract state model from its materialization in an OCEL or graph; and it defines an integrated analysis and evaluation agenda.

Table 1 positions representative related approaches using dimensions that matter for this paper. The comparison concerns the abstractions constructed directly by each work stream; it does not exclude combinations with the state layer.

The table shows that SA-OCPM is not intended to replace object-centric variants, local process models, conformance analysis, or root-cause methods. It provides a state-centered layer that can organize and feed these analyses. Its distinctive combination is a temporal state abstraction tied to a selected object type, support for both specified and discovered states, explicit state transitions, and analysis of both persistence and change while preserving object context.

Figure 3 makes the distinction between three analytical perspectives explicit. Object-centric variants group executions by graph structure and selected labels; equivalence can also be defined over attributes [Adams et al., 2022, 2024]. Here, *local process executions* refers to selected execution fragments, for example those delimited by object relations, events, or time intervals in lifecycle segmentation [Berti et al., 2026b]. Such fragments support local analysis, and object-centric local process models summarize recurring behavior without prescribing an operational state calendar [Peeva et al., 2025]. SA-OCPM supplies the additional temporal semantics: the same structural behavior can be distinguished by the condition in which it occurs, and state changes determine episode and transition contexts. This enables state-specific dwell, recovery, and intra-state versus inter-state pattern analysis [Kretzschmann et al., 2026a,b]. Variants and local methods can consume these annotations, but their extraction or grouping criteria alone do not construct them.

Table 1: Positioning of representative work streams relative to the state-aware framework.

Work stream	Role of state	Primary unit	Object context	Difference from this paper
OCDFG conformance and performance	State is not a first-class temporal abstraction	Typed directly-follows edges	Preserved by edge types	Does not define state construction, calendars, or state-determined patterns
Object-centric variants and super variants	May use attributes for grouping	Complete or selected executions	Preserved	Focuses on execution comparison rather than persistence and state change
Object-centric local process models	Not the segmentation principle	Recurring local behavior	Preserved in object-centric models	Patterns are not anchored to operational state episodes or boundaries
Data-aware OCPM	Data conditions explain behavior	Events, decisions, and attributes	Preserved	Does not necessarily produce a finite, temporally continuous state calendar
Original SA-OCPM	Explicit transitions and state-aware events	Enriched OCEL and graph	Preserved	Introduces the core idea but not the complete unified framework
Automatic execution-state abstraction	States learned from lifecycle windows	Windows and SOM cells	Encoded in features	Focuses on discovery and boundary conditions
State-aware pattern mining	Episodes determine local segments	Intra-state and inter-state graphs	Preserved in local graphs	Focuses on pattern aggregation after states exist
This paper	Versioned manual or learned state notion	Incidences, calendars, graphs, episodes, patterns, outcomes	Preserved explicitly	Unifies construction, representation, analysis, tool support, and evaluation

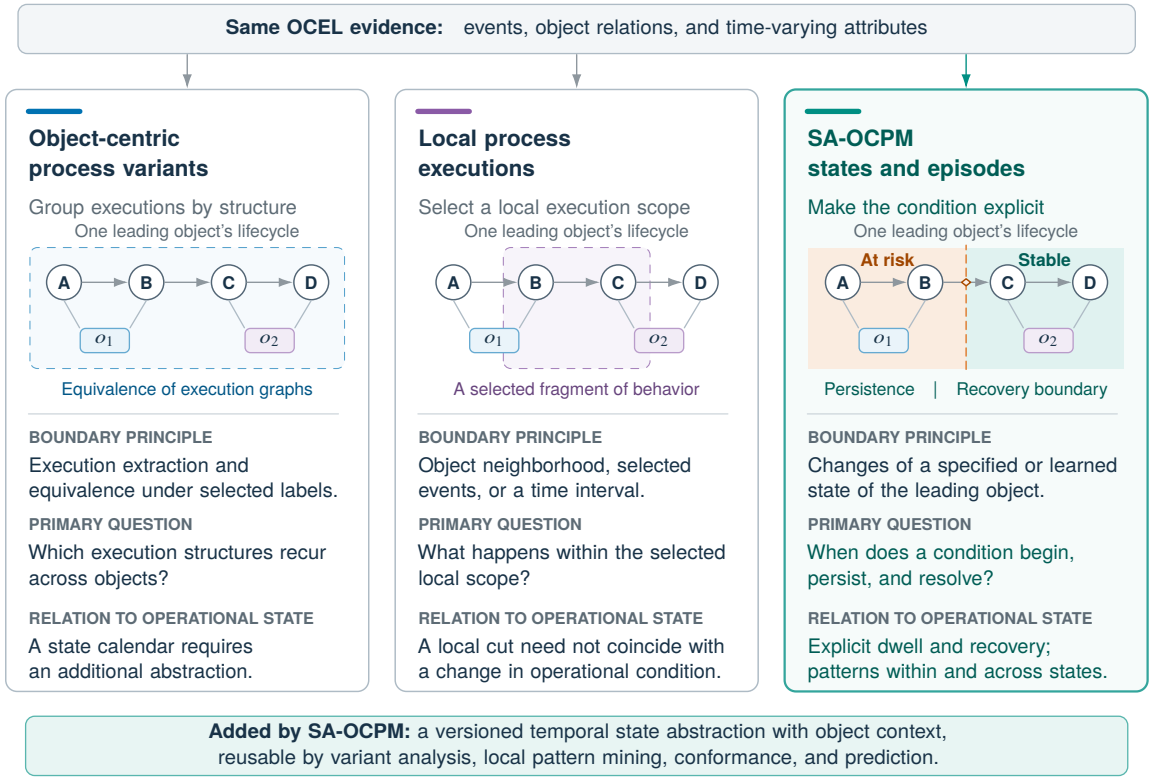

Fig. 3 Conceptual comparison of object-centric process variants, local process executions, and SA-OCPM states. The schematic repeats one leading object's activity sequence and its context-object links; outlines indicate structural or local scope, while the state calendar supplies operational boundaries. The comparison concerns the native analytical focus of each perspective. SA-OCPM contributes explicit state semantics that variants and local analyses can reuse.

3 Preliminaries

This section defines the event-data and graph concepts on which SA-OCPM is built. No state concept is used yet. Table 2 summarizes the main symbols. The definitions are intentionally close to OCEL 2.0 while abstracting from a particular serialization format.

3.1 Object-centric event logs

Let T be a totally ordered time domain, represented on a common real-valued time scale whenever durations are computed. A concrete log fixes sets $E, O, A, OT, K_E, K_O, Q, V$ of event identifiers, object identifiers, activities, object types, event-attribute names, object-attribute names, qualifiers, and values. All of these sets are finite, except that the value set V, like the time domain T, may be infinite.

Definition 1 (Object-centric event log). An object-centric event log is a tuple

$$L = (E, O, A, OT, K_E, K_O, Q, V, \text{act}, \text{time}, \text{type}, R_{EO}, R_{OO}, \text{val}_E, \text{upd}_O, <_L),$$

where the components satisfy the following conditions. E is a finite set of events, O is a finite set of objects, A, OT, K_E, K_O, and Q are finite sets of activities, object types, event-attribute names, object-attribute names, and qualifiers, respectively, and V is a set of values. The function $\text{act} : E \to A$ assigns an activity to every event, $\text{time} : E \to T$ assigns a timestamp, and $\text{type} : O \to OT$ assigns an object type. The qualified event-to-object relation $R_{EO} \subseteq E \times Q \times O$. The qualified object-to-object relation $R_{OO} \subseteq O \times Q \times O$. The partial function $\text{val}_E : E \times K_E \nrightarrow V$ assigns event-attribute values. The finite set $\text{upd}_O \subseteq O \times K_O \times T \times V$ records timestamped object-attribute updates. Finally, $<_L$ is a strict total order on E that is compatible with time: $\text{time}(e) < \text{time}(e')$ implies $e <_L e'$. It resolves ties when events have equal timestamps.

Definition 1 separates object-attribute updates from their time-valid values. For an object o, attribute k, and time t, let $\text{hist}_L(o, k, t)$ be the updates (o, k, t', v) in upd_O with $t' \le t$. If this set is nonempty, the time-valid value $\text{val}_O^L(o, k, t)$ is the value of its latest update. We require at most one value for each (o, k, t'); conflicting simultaneous updates must be resolved during preprocessing and the resolution recorded. If the history is empty, the value is undefined. Static initial values are represented by updates effective no later than the observation start. This abstraction captures OCEL 2.0 object histories while omitting serialization-specific attribute declarations.

The relation R_{OO} is static: OCEL 2.0 does not by itself record when an object-to-object relation became known or ceased to hold. A retrospective context may use it as given. Monitoring or prediction must instead restrict relational information to links known at the decision point, established from source metadata or a declared reconstruction. The presence of a relation in the completed log is not evidence of its earlier availability.

The qualifiers in R_{EO} and R_{OO} can distinguish roles such as created, consumed, delivered-by, component-of, or predecessor. SA-OCPM can retain qualifiers in local graphs and expressions, but the basic definitions below require only the existence of a relation. For an event e, its related objects are

$$\text{rel}_L(e) = \{o \in O \mid \exists q \in Q : (e, q, o) \in R_{EO}\}.$$

For an object o, its related events are

$$\text{ev}_L(o) = \{e \in E \mid \exists q \in Q : (e, q, o) \in R_{EO}\}.$$

Figure 4 visualizes the OCEL data model of Definition 1. It separates event and object entities from their types and attributes, represents qualified event-to-object and object-to-object relations explicitly, and shows object attributes as timestamped histories. The illustrative fragment emphasizes both many-to-many event participation and time-valid object values.

Table 2: Main notation used in the formalization.

Symbol	Meaning	Notes
L	Base object-centric event log	Contains events, objects, attributes, relations, updates, and event order
E, O	Finite event and object sets	An event may relate to several objects
A, OT	Activities and object types	τ denotes a selected leading object type
R_{EO}, R_{OO}	Qualified event-to-object and object-to-object relations	Qualifiers can be retained in expressions and local graphs
val_E, upd_O	Event values and time-stamped object-attribute updates	val_O^L resolves the latest valid object value
$\text{life}_L(o)$	Lifecycle of object o	Related events ordered by the log order
G_L	Object-centric directly-follows graph	Edges are typed by object type
M_τ	State notion for leading type τ	Contains state set, context extractor, assignment, and provenance
σ_M	Induced state function	Maps a leading object and observation point to a finite state
$L_M^{\text{SA}}, G_M^{\text{SA}}$	State-aware OCEL and SA-OCDFG	Base log retained; graph recovery requires lifecycle provenance
Δ_M	Explicit state-change records	Includes time or point, source, target, and optional cause
H_z, $\text{sig}(H_z)$	Segment graph and structural signature	Used to aggregate recurring state-aware patterns

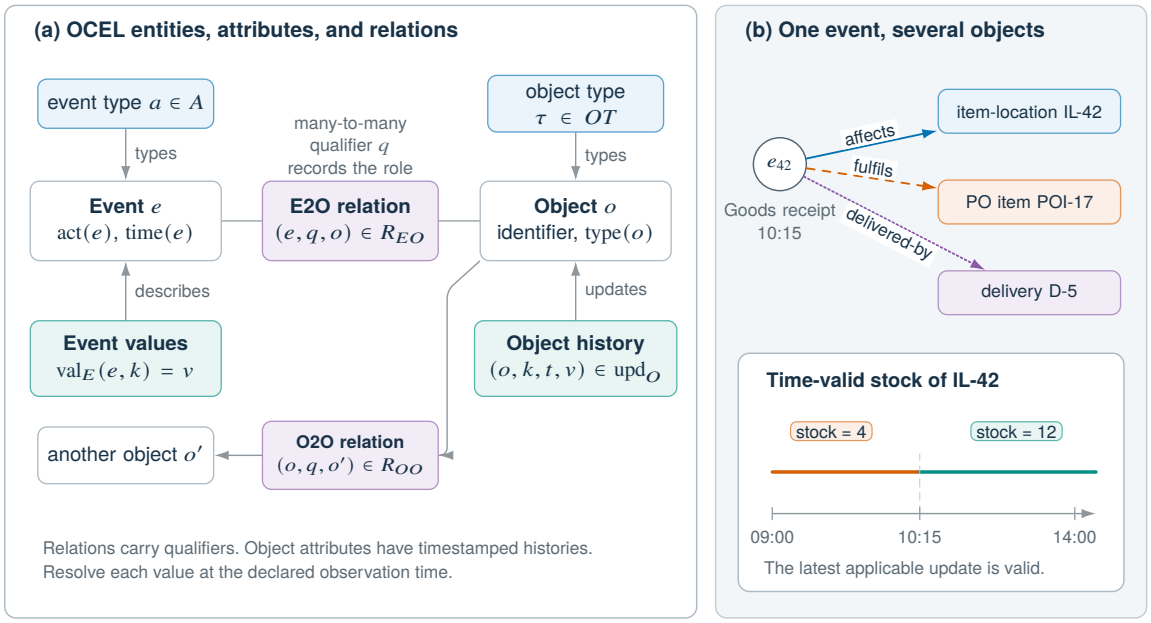

Fig. 4 OCEL 2.0 data model used in the formalization. Events and objects have types and attributes; event-to-object and object-to-object relations carry qualifiers; and object attributes are represented by timestamped updates. The fragment on the right illustrates a shared event and a time-varying object value.

3.2 Object lifecycles and projections

For an object type $\tau \in OT$, define $O_\tau = \{o \in O \mid \mathrm{type}(o) = \tau\}$. The behavior observed from the perspective of one object is its lifecycle.

Definition 2 (Object lifecycle). Let o be an object. The lifecycle of o in L is the sequence

$$\mathrm{life}_L(o) = \langle e_1, e_2, \ldots, e_n \rangle$$

that contains exactly the events in $\mathrm{ev}_L(o)$ and is ordered by $<_L$. Thus, $e_i <_L e_{i+1}$ for all $1 \le i < n$. The activity projection is $\mathrm{actlife}_L(o) = \langle \mathrm{act}(e_1), \ldots, \mathrm{act}(e_n) \rangle$.

A lifecycle is a perspective, not an isolated case. Event e_i can also occur in the lifecycles of other objects, possibly of other types. This shared-event semantics is what preserves synchronization. For example, one Goods Receipt event may occur in the lifecycles of a delivery, several purchase order items, and several material-location objects.

For a selected object type τ, an object-type projection can be formed by collecting the lifecycles $\mathrm{life}_L(o)$ for all o in O_τ. Unlike conventional flattening, this projection need not duplicate or discard the underlying event. It merely states which lifecycle orders are used for an analysis. Later, τ will serve as the leading object type whose state is modeled.

A lifecycle may contain long periods without events. Duration-based analyses therefore use both event order and time. If two consecutive related events e_i and e_{i+1} occur at times t_i and t_{i+1}, an event-derived condition observed after e_i is commonly treated as valid on $[t_i, t_{i+1})$. Object-attribute updates between those events can refine this interval. Section 4.2 makes the observation points and validity policy explicit for state notions.

3.3 Object-centric directly-follows graphs

A directly-follows relation states that one activity immediately follows another in a selected sequence. In OCPM, the relation is typed by the object lifecycle that induces it. Add two auxiliary labels START_τ and END_τ for every object type τ. All auxiliary labels, state-aware activity tuples, and ordinary activity labels belong to disjoint tagged namespaces, so an activity name cannot collide with a boundary or change marker.

For object o in O_τ with activity lifecycle $\langle a_1, \ldots, a_n \rangle$, define the augmented sequence

$$\mathrm{aug}_\tau(o) = \langle \mathsf{START}_\tau, a_1, \ldots, a_n, \mathsf{END}_\tau \rangle.$$

If $n = 0$, the sequence contains only START_τ and END_τ. Let $\mathrm{adj}_\tau(o, x, y)$ count the positions at which label x is directly followed by label y in $\mathrm{aug}_\tau(o)$.

Definition 3 (Object-centric directly-follows graph). The OCDFG of L is a typed weighted directed graph

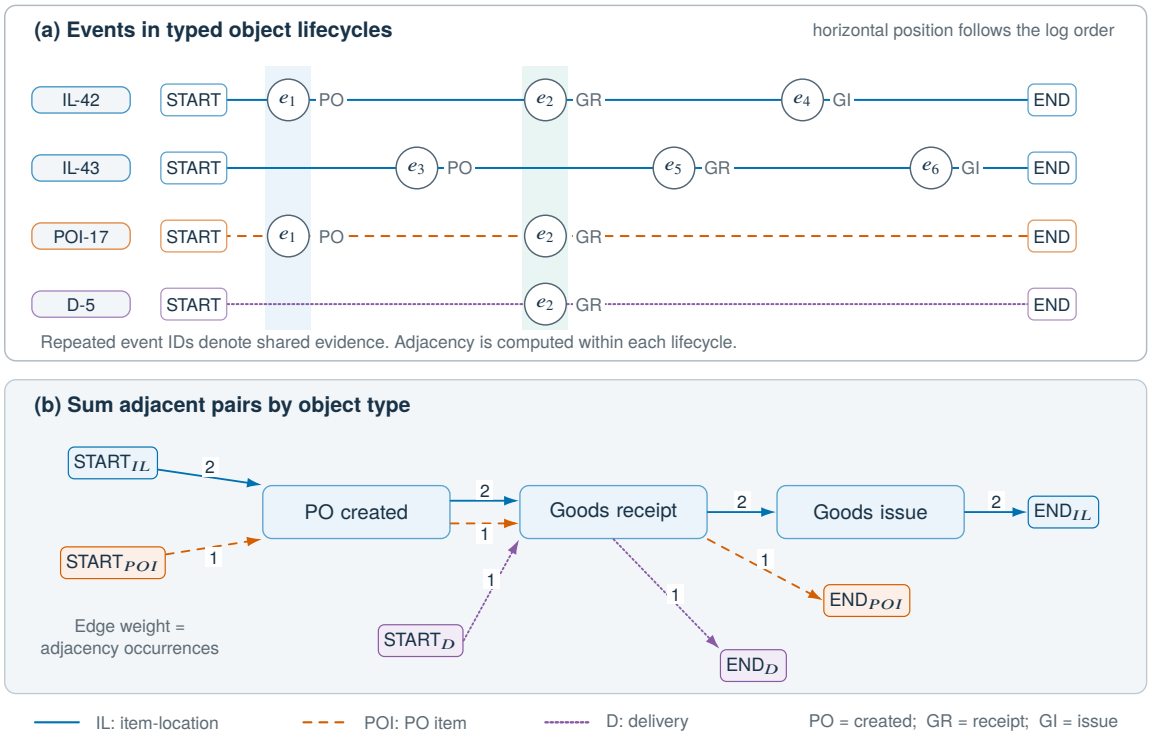

Fig. 5 Construction of an object-centric directly-follows graph. Shared events occur in multiple typed object lifecycles, each lifecycle is augmented with typed START and END labels, and adjacent activity pairs are summed into typed weighted edges.

$$G_L = (N, E_{DF}, w),$$

where N contains the activity labels in A that occur in L and the labels START_τ and END_τ for every object type $\tau \in OT$ with $O_\tau \neq \emptyset$. For labels x and y and object type τ, define

$$w(x, y, \tau) = \sum_{o \in O_\tau} \mathrm{adj}_\tau(o, x, y).$$

A typed edge (x, y, τ) is in E_{DF} if and only if $w(x, y, \tau) > 0$. The edge weight $w(x, y, \tau)$ is the number of directly-follows occurrences induced by objects of type τ. Edges with the same endpoints but different object types remain distinct.

An OCDFG can also carry node frequencies, object support, waiting-time summaries, and conformance annotations. Definition 3 retains only what is needed for the state-aware extension. In particular, it makes clear that an edge is produced by adjacency in an object lifecycle and not simply by global timestamp order.

Figure 5 illustrates the construction from an OCEL fragment to an OCDFG. Shared event identifiers appear in several typed object lifecycles, START and END labels augment each lifecycle, and adjacent pairs are accumulated by object type. The resulting graph deliberately omits state labels and establishes the baseline representation used by the state-aware extension.

The absence of state in Definition 3 has two consequences. First, all occurrences of an activity are merged even when they happen under different operational conditions. Second, a transition from one condition to another is visible only indirectly through attributes or surrounding behavior. The next section adds a state layer while preserving the underlying OCEL and its typed lifecycle semantics.

4 Approach

4.1 Overview and design principles

SA-OCPM starts from a leading object type τ. The leading type identifies the objects whose operational state organizes the analysis. Other object types remain essential context. In inventory management, the leading object may be a material-location or item-location object, while purchase order items, sales order items, deliveries,

suppliers, and warehouses explain what happens around it. In manufacturing, the leading object may be a machine, while production orders, work orders, components, alarms, and operators provide context.

The leading-object choice prevents an ambiguous phrase such as "the state of the event." Events do not generally have one intrinsic operational state. An event related to two item-location objects can occur while one is `Understock` and the other is `Overstock`. The primary unit of state annotation is therefore the event-object incidence (e, o), not the event alone. Event-level state attributes remain useful materializations for tools and formats, but they require an explicit aggregation or projection policy.

The framework follows seven design principles. First, it is a conservative extension: the original OCEL remains recoverable. Second, a state notion is objective-relative and associated with a leading object type. Third, the method used to construct a state is separated from the analyses that consume it. Fourth, time validity and same-timestamp ordering are explicit. Fifth, multi-object ambiguity is preserved or resolved by a declared policy. Sixth, every state notion has provenance, including its version, parameters, and feature or expression definition. Seventh, state validity is evaluated through coverage, stability, interpretability, and usefulness rather than assumed.

Figure 6 provides the overview of the approach. The expression-based branch derives auditable assignments from domain objectives, time-valid context, and ordered rules. The automatic branch derives recurrent regimes from lifecycle windows, feature encoding, PCA, and a self-organizing map. Both instantiate the same versioned state-notion interface, feed the same state-aware representations and analyses, and participate in an expert refinement loop.

4.1.1 Selecting the leading object type

The leading object type should identify the entity whose condition, exposure, or recovery the analysis is intended to explain. A useful starting question is: "Which entity should remain identifiable from entry into an undesirable condition until recovery, and for which entity could an operational response be assigned?" The answer need not be the most frequent object type or the case identifier used in an existing report. We recommend the following procedure.

1. **Start from the objective and the unit of action.** For local stock availability, choose an item-location rather than a purchase order: the stock condition persists across many orders. For equipment downtime, choose the machine rather than the maintenance work order that records one response. If the objective changes to fulfillment delay or repair-workflow compliance, an order or work order may instead be appropriate.

2. **Choose a granularity with coherent state semantics.** One leading object should admit one state under the

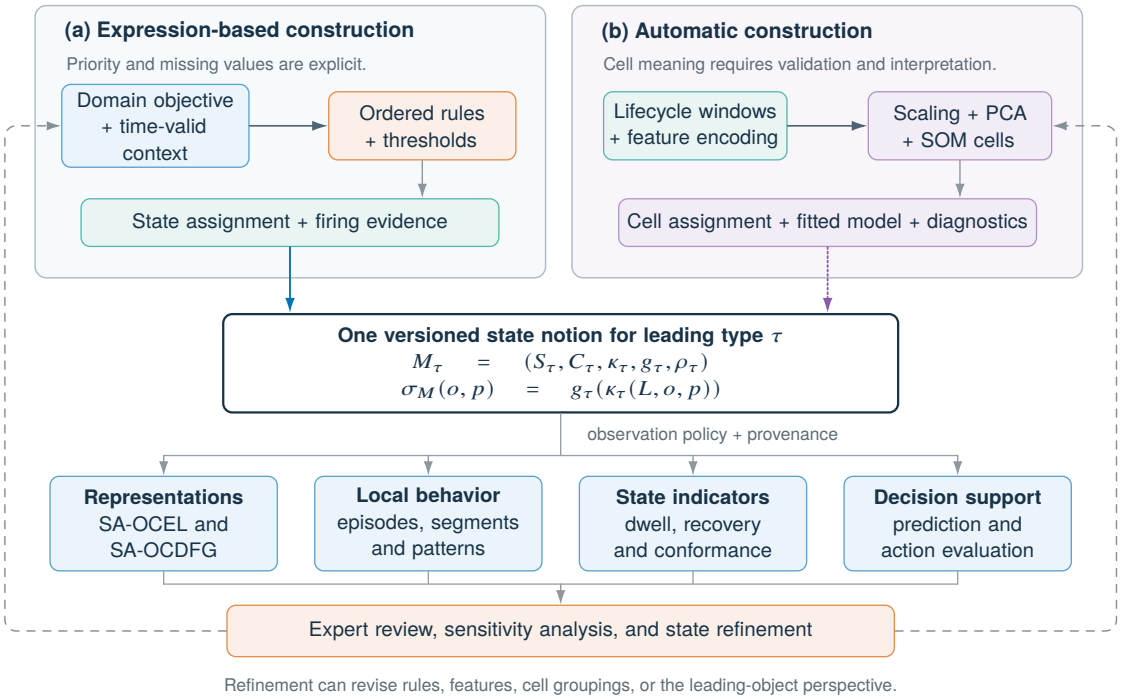

Fig. 6 Complementary routes to a state notion. Expression-based construction and automatic lifecycle-window abstraction instantiate the same versioned interface and therefore feed the same SA-OCEL, SA-OCDFG, pattern, indicator, prediction, and decision-support analyses. Expert review closes the refinement loop.

selected notion at each observation point. A material aggregated across warehouses may be understocked at one location and overstocked at another; an item-location object resolves that ambiguity. Likewise, a care episode may be preferable to a patient identifier when simultaneous treatments require distinct states. If the needed composite entity is absent from the OCEL, define its identity and relations explicitly during data preparation and retain the source objects.

3. **Check observability and lifecycle coverage.** Confirm that relevant events, time-valid attributes, and related objects can be associated with each candidate over the intended horizon. Inspect whether entry, persistence, and exit are observable, how long gaps last, and how much evidence would yield `Unknown`. Long lifecycles justify state-based segmentation; sparse or very short lifecycles can make dwell, recurrence, or learned-window analyses unreliable. A short-lived object is still suitable when it matches the question and its relevant condition is observable.

4. **Pilot plausible alternatives and record the choice.** On a representative sample, compare missing evidence, episode interpretability, transition frequency, object-context completeness, and the stability of the intended result. For learned states, also check window support and transfer across objects. Prefer the least aggregated type that answers the operational question with adequate evidence; retain separate perspectives if different questions require different types. Record the objective, selected type, identity and relation rules, horizon, and rationale in the state notion's provenance. Use training or exploratory data for this choice before evaluating on held-out data.

Selecting a leading type establishes the reference entity for state; it does not discard the other object types. Events shared by several leading objects retain separate incidence-level assignments. Related suppliers, clinicians, components, or orders can therefore explain a state without becoming the entity whose dwell or recovery is measured.

4.2 State notions

A state notion requires an ordered observation axis and a finite observation horizon $[T_o^0, T_o^1)$, with $T_o^0 < T_o^1$, for each leading object o. All lifecycle timestamps lie in the closed interval from T_o^0 to T_o^1; endpoint observations have zero remaining exposure. Restricting the event population first produces a declared sublog. The finite sequence $P_L(o) = \langle p_0, \ldots, p_m \rangle$ starts at an initial point p_0 at T_o^0, includes pre-event and post-event query points for every lifecycle event, and includes the declared state-relevant updates and sampling points up to T_o^1. Objects without events still have an initial point. A terminal cut b_{end}, after all these points at T_o^1, closes the horizon.

Write $\text{ts}(p)$ for the timestamp of an observation point or terminal cut. Observation points are totally ordered by timestamp and a deterministic order within each timestamp, respecting $<_L$ and the declared placement of updates relative to events. Their information prefixes, initial context, and evaluation schedule are part of the state notion's provenance; dependence of $P_L(o)$ on that notion is suppressed in the notation. State is carried forward between evaluation points. A pre-event query reads that carried state and does not itself clear event context or trigger a new assignment. Post-event evaluation incorporates the information made available by processing that event. A time-dependent expression requires its relevant threshold crossings to be evaluation points; a regular sampling schedule instead defines a sampled approximation, whose resolution must be reported.

Definition 4 (State notion). Let τ be a leading object type. A state notion for τ is a tuple

$$M_\tau = (S_\tau, C_\tau, \kappa_\tau, g_\tau, \rho_\tau),$$

where S_τ is a nonempty finite set of state labels, C_τ is a context space, κ_τ maps a log L, a leading object $o \in O_\tau$, and an observation point $p \in P_L(o)$ to a context $c \in C_\tau$, and $g_\tau : C_\tau \to S_\tau$ maps the context to a state. The provenance descriptor ρ_τ records at least an identifier, version, construction mode, parameters, attributes or features used, horizon, initial-context policy, observation schedule, and temporal ordering and availability policies. The induced state function is

$$\sigma_M(o, p) = g_\tau(\kappa_\tau(L, o, p)).$$

The log L under analysis is fixed and therefore suppressed in the notation σ_M. In subscripts, M abbreviates M_τ when the leading type is clear from the context.

The finite state set may include an explicit `Unknown` state. The state function is total on the declared observation axis; insufficient evidence must receive an explicit state rather than silently remove lifecycle regions. `Unknown` should not be confused with `Normal`. It states that the current evidence or rule coverage is insufficient.

Table 3: Comparison of expression-based, automatic, and hybrid state construction.

Dimension	Expression-based	Automatic	Hybrid use
Source of semantics	Domain rules, policies, thresholds, and derived quantities	Recurring lifecycle-window structure and feature similarity	Use learned regimes to refine or challenge rules
Clarity	Direct branch-level explanation	Requires cell, feature, and boundary interpretation	Translate stable cells into auditable rules when appropriate
Adaptability	Changes only when rules or parameters change	Can reveal previously unknown regimes	Retain both taxonomies for different objectives
Main risks	Incorrect thresholds, overlap, missing values, policy drift	Leakage, unstable granularity, sparse cells, opaque features	Inconsistent mapping or unjustified merging
Validation	Coverage, conflict, boundary, and sensitivity checks	Occupancy, stability, quality, temporal validity, and expert labels	Alignment plus incremental analytical utility
Best suited for	Monitoring, conformance, auditable deployment	Exploration, hypothesis generation, regime discovery	Iterative discovery-to-deployment workflow

For an event e_i in $\text{life}_L(o)$, let p_i^- and p_i^+ be its pre-event and post-event query points under the declared tie policy. In the event-based special case, the pre-state equals the preceding event's post-state, or the initial state for $i = 1$. The preceding post-event point and the current pre-event point can have different timestamps; the pre-event and post-event queries for a given event use that event's timestamp. The pre-state and post-state are

$$s_M^-(e_i, o) = \sigma_M(o, p_i^-), \qquad s_M^+(e_i, o) = \sigma_M(o, p_i^+).$$

A state-changing incidence satisfies $s_M^-(e_i, o) \neq s_M^+(e_i, o)$. If an object-attribute update changes the state between two related events, the change is associated with its own observation point rather than falsely attributed to either event. This distinction is useful when a state change is exogenous to the recorded activity.

The state calendar of object o is the maximal piecewise-constant representation of σ_M along $P_L(o)$. It can be written as

$$\text{cal}_M(o) = \langle (b_1, s_1), \ldots, (b_r, s_r) \rangle,$$

where $b_1 = p_0$, b_j is the beginning observation point of state s_j, and adjacent states differ. Set $b_{r+1} = b_{\text{end}}$. Each block covers the ordered points from b_j up to, but excluding, b_{j+1}, and induces a time interval $[\text{ts}(b_j), \text{ts}(b_{j+1}))$. Distinct blocks can begin at the same timestamp: such intermediate states retain their ordered transitions but have zero duration. At a physical time $t < T_o^1$, the time-state is the state of the last calendar boundary with timestamp at most t. Thus the nonempty intervals partition the horizon. No persistence beyond T_o^1 is inferred.

Definition 5 (Well-formed state notion). A state notion M_τ is well formed for L if it is deterministic for the same input and provenance, returns a state for every observation point of every object in O_τ, uses only information permitted by its declared temporal policy, and yields a finite state calendar. A learned state notion must additionally store the fitted preprocessing and quantization parameters needed to reproduce assignments.

Well-formedness does not imply domain validity. A deterministic rule can encode a poor threshold, and a reproducible clustering can discover unstable or uninterpretable cells. Validity is assessed later through expert agreement, predictive or diagnostic utility, stability under perturbation, and sensitivity to state-construction choices.

Table 3 contrasts expression-based and automatically detected state notions. The two modes differ in how κ_τ and g_τ are obtained, but they share the same induced state function and can therefore feed the same SA-OCPM analyses.

4.3 Expression-based state notions

An expression-based state notion is appropriate when operational semantics are known or can be elicited. Examples include inventory regimes defined by a safety stock and maximum level, order states defined by blocking and fulfillment attributes, patient-risk levels defined by clinical thresholds, and machine conditions defined by mode and alarm combinations.

For an event-object incidence (e, o), the expression context can contain the event identifier, activity, time, event attributes, object identifier and type, time-valid object attributes, counts or properties of related objects, and derived values. A derived value can be a stock balance reconstructed from preceding movements, elapsed time since the last maintenance action, number of open purchase orders, or a rolling sensor statistic. The derivation itself becomes part of the provenance.

Definition 6 (Time-valid expression context). For leading object o at an evaluation point p, the expression context $c_{\text{expr}}(L, o, p)$ is a partial valuation over named fields. Event fields refer to the event processed at p,

when one exists; their treatment at update-only or sampling points must be declared, for example as undefined. At a query point without evaluation, the previous evaluation context is carried forward. Object field `object.k` resolves to the latest update of attribute k admitted by the information prefix at p. This agrees with val_O^L after all updates at the timestamp have been processed, but can differ at an earlier phase of that timestamp. Relational fields use only the objects and links admitted by the temporal policy. A comparison with an undefined operand evaluates to false unless the expression explicitly tests for missingness. Consequently, its Boolean negation evaluates to true. This is two-valued expression semantics, not SQL's three-valued NULL semantics; rules that distinguish missing evidence must test missingness explicitly.

An ordered CASE definition contains predicates $P_1, \ldots, P_m$, result expressions $r_1, \ldots, r_m$, and a default r_0. Predicates can use comparisons, membership tests, pattern matching, null tests, and Boolean composition.

Definition 7 (Expression-based state assignment). Let

$$D = \text{CASE WHEN } P_1 \text{ THEN } r_1 \cdots \text{WHEN } P_m \text{ THEN } r_m \text{ ELSE } r_0 \text{ END}.$$

For context c, let j be the smallest index for which $P_j(c)$ is true. Then $g_D(c) = r_j(c)$. If no predicate is true, $g_D(c) = r_0(c)$. The result is converted to a member of the declared finite state set S_τ, or to `Unknown` if the result is missing or outside the declared set. A definition that can produce `Unknown` must therefore include `Unknown` in S_τ.

The ordered semantics makes priority explicit. For example, a blocked order can be assigned `Blocked` even if it also satisfies a high-value condition by placing the block predicate first. This is preferable to overlapping, unordered rules whose resolution depends on implementation details.

A simple inventory definition for item-location object o at time t, with all three numeric inputs defined and $\mathrm{safety}(o,t) \le \mathrm{maximum}(o,t)$, is

$$\mathrm{state}(o,t) = \begin{cases} \texttt{Understock}, & \mathrm{stock}(o,t) < \mathrm{safety}(o,t), \\ \texttt{Overstock}, & \mathrm{stock}(o,t) > \mathrm{maximum}(o,t), \\ \texttt{Normal}, & \text{otherwise.} \end{cases}$$

Outside these assumptions, a preceding `Unknown` branch must handle missing stock, missing thresholds, and inconsistent threshold ordering. A `Critical Understock` branch can additionally test whether confirmed demand exceeds available and inbound stock. The thresholds can be static object attributes, time-varying policy attributes, or values reconstructed from another system. Each assignment can retain the firing predicate and the values used.

The multi-object case requires care. Suppose event e is related to two leading objects o_1 and o_2. The incidence states $s_M^+(e, o_1)$ and $s_M^+(e, o_2)$, and likewise the pre-states, should normally be retained separately. If a consumer requires one event-level state, four policies are possible. A single-object policy assigns a state only when exactly one leading object is related. An explode policy creates one event perspective per related leading object. A set-valued policy stores all distinct states. An aggregation policy applies a declared operator, such as the most severe state, a priority rule, or existential satisfaction. No aggregation is universally correct.

Expression definitions should be validated before analysis. Coverage reports show how often each branch fires and how much time is assigned `Unknown`. Conflict checks identify predicates that overlap before priority is applied. Boundary inspection shows objects just above and below a threshold. Sensitivity analysis varies thresholds and checks whether key findings remain. These steps are especially important when states will be used for conformance or intervention decisions.

4.4 Automatically detected state notions

Automatic state construction is useful when no accepted taxonomy exists, when a known status attribute is too coarse, or when the operational regime depends on a combination of behavior and context. The goal is not to replace expertise with arbitrary clusters. It is to expose recurrent execution conditions, organize their transitions, and provide evidence that allows analysts to interpret and refine them.

Let o be a leading object with lifecycle $\langle e_1, \ldots, e_n \rangle$. Define an event-object feature encoding $\phi(e_i, o) \in \mathbb{R}^d$. The encoding can include one-hot activity indicators, counts of related objects by type or qualifier, selected numeric event and object attributes, one-hot categorical attributes, elapsed-time features, and derived context. All feature choices, missing-value rules, scaling parameters, and category dictionaries are stored in ρ_τ.

Definition 8 (Event-object feature encoding). An encoding ϕ is temporally admissible if $\phi(e_i, o)$ uses only information permitted at the observation point associated with e_i. It is perspective-consistent if object-level features refer to o and relational features are defined relative to o. The encoded sequence of o is

$$\Phi_L(o) = \langle \phi(e_1, o), \ldots, \phi(e_n, o) \rangle.$$

Automatic state discovery should avoid target leakage. A future event, a future object attribute, or the final outcome must not be included when the discovered states are intended for monitoring or prediction. For retrospective descriptive analysis, a wider context may be legitimate, but it must be declared.

A state often reflects a short history rather than one event. Let $w \geq 1$ be an integer window length measured in lifecycle events. For $w \leq i \leq n$, concatenate or aggregate the last w encodings.

Definition 9 (Lifecycle window). The window representation ending at e_i is

$$z^w_{o,i} = \text{concat}(\phi(e_{i-w+1}, o), \ldots, \phi(e_i, o)) \,.$$

For early positions, the implementation may omit incomplete windows, pad them, or use shorter windows. The policy must be fixed. Aggregated windows, such as activity counts and summary statistics over the last w events, are an alternative to concatenation and can handle variable-length time windows.

Let R be a fitted dimensionality-reduction map, such as PCA, and let q be a fitted finite quantizer or clustering map. The automatic state is

$$\sigma_{\text{auto}}(o, i) = q\Big(R(z^w_{o,i})\Big) \,.$$

This instantiates the induced state function of Definition 4 at the post-event point of e_i. For omitted early windows, the initial point, and objects with fewer than w events, a dedicated `Warm-Up` or `Unknown` state ensures totality. Between complete windows, the last assignment is carried forward unless an additional update rule is declared. Padding and shortening require an explicit encoding into the fixed input dimension of R. Shortening the analyzed log is a separate projection and cannot be treated as preserving all incidences of the original log.

In the SOM-based instantiation, R maps standardized window vectors to a low-dimensional PCA space, and q assigns a nearest prototype on a two-dimensional grid. Equal-distance ties use a fixed cell order. Consecutive windows assigned to different cells produce a state transition. The grid aims to preserve neighborhood structure; similarity of adjacent cells is a diagnostic to assess rather than a guaranteed property. Prospective evaluation fits scaling, feature dictionaries, PCA, and SOM parameters only on training data and freezes them before transforming held-out windows. The permitted information prefix also applies to features of related objects.

The window length controls temporal resolution. A very short window can make states react to individual activities and can produce unstable switching. A very long window can hide important boundaries and can delay detection. One selection strategy evaluates PCA reconstruction error across candidate window lengths and looks for an elbow beyond which additional history yields limited representational gain [Berti et al., 2026c]. Domain-relevant durations and stability of the resulting state calendars should also be considered.

The SOM grid size controls granularity. A small grid merges distinct regimes. A large grid produces sparse cells and fragments behavior. Useful diagnostics include occupied-cell coverage, normalized occupancy entropy, quantization error, and preservation of local transitions. Let G_{occ} be the set of occupied cells and, for $g \in G_{\text{occ}}$, let p_g be the fraction of assigned windows in cell g. For $|G_{\text{occ}}| \geq 2$, a normalized occupancy entropy can be defined as

$$H = -\frac{\sum_{g \in G_{\text{occ}}} p_g \log p_g}{\log |G_{\text{occ}}|}.$$

For one occupied cell, define $H = 0$; with no assigned windows, occupancy diagnostics are undefined. A simple transition-neighborhood score is the fraction of consecutive windows within the same object whose assigned cells are identical or adjacent on the grid, provided at least one such pair exists. The grid adjacency convention must be fixed. Candidate grids can be compared using these measures and expert interpretability rather than one universal optimum.

A learned cell initially has an identifier, not a business meaning. Interpretation combines several views. Density shows how common the state is. Attribute overlays show distributions of stock, status, risk, cost, or machine measurements. The cell-specific DFG shows behavior while assigned to the cell. Entry and exit windows reveal boundary conditions. Leading features can be contrasted with stable windows from the same or neighboring cells. Analysts can then name a cell, merge similar cells, split a heterogeneous cell through another model, or translate the finding into an expression-based rule.

This hybrid workflow is important. Automatic states are especially valuable for discovery, while expression-based states are often preferable for deployment because they are stable, auditable, and easy to communicate. Conversely, learned states can reveal that a manually defined `Normal` state contains several behaviorally different

regimes. The framework permits both state notions to be retained and compared rather than forcing one to replace the other.

4.5 State-aware object-centric event logs

The state notion now has to be attached to the event data. The primary annotation is defined over event-object incidences for the leading type.

Definition 10 (State-annotated incidence). For leading type τ and state notion M_τ, the set of leading incidences is

$$I_\tau(L) = \{(e, o) \mid o \in O_\tau \wedge \exists q : (e, q, o) \in R_{EO}\}.$$

For incidence (e, o), the annotation is

$$\mathrm{ann}_M(e, o) = \big(s^-_M(e, o), s^+_M(e, o), \mathrm{conf}_M(e, o), \mathrm{prov}_M\big),$$

where the first two components are the pre-state and post-state, conf_M is an optional confidence or evidence record, and prov_M refers to the state-notion provenance.

For a deterministic expression, confidence may be omitted or may record the firing branch and input values. For a learned model, confidence can contain the distance to the best matching prototype, the margin to the next prototype, or a stability score across model fits. The annotation relation preserves different states for different leading objects related to the same event.

Definition 11 (State-aware object-centric event log). A state-aware object-centric event log is the tuple

$$L^{\mathrm{SA}}_M = (L, \tau, M_\tau, I_\tau(L), \mathrm{ann}_M, \Delta_M, \pi_M),$$

where L is the base OCEL, Δ_M is the set of explicit state-change records, and π_M is a materialization policy. A state-change record has the form

$$\delta = (o, p, s, s', \mathit{cause}, \mathit{evidence})$$

with $o \in O_\tau$, $p \in P_L(o)$, and distinct $s, s' \in S_\tau$. It states that o changes from s to s' at p. For each object, the records correspond one-to-one to its internal calendar boundaries $b_2, \ldots, b_r$, including boundaries with equal timestamps: the record at b_j has source s_{j-1} and target s_j. Initialization at b_1 and censoring at b_{end} are not changes. The optional cause identifies the triggering event, update, or sampling tick; absent evidence of a trigger, it is unknown. Such a link records provenance rather than establishing a causal effect. Dropping the added components recovers L exactly.

The conservative structure is useful for comparison and revision. Analysts can construct several SA-OCEL views over the same base log, such as a stock-regime view and a fulfillment-risk view. Replacing a state notion does not require overwriting activity names or losing the original attributes.

A concrete OCEL file does not necessarily support incidence-level derived attributes. The materialization policy π_M maps the abstract annotations into a representation that a selected tool or exchange format can consume. Table 4 lists four annotation policies and a complementary transition-coalescing policy. Recoverability of the abstract tuple does not imply that every materialized export preserves its information.

The explode materialization preserves one state per leading-object perspective by duplicating a derived analysis event or by introducing a perspective object. It is semantically clean but can enlarge the data. The set-valued materialization preserves ambiguity in one event attribute but is not supported by every algorithm. A

Table 4: Materialization policies for incidence-level state annotations.

Policy	Concrete representation	What it preserves	Main risk or use
Single-leading-object restriction	Assign an event state when exactly one leading object is related	Unambiguous event-level state and original event count	Events with zero or multiple leading objects remain unannotated
Explode by perspective	Create one analysis occurrence per event-leading-object incidence	Every incidence state and its object identity	Can enlarge the data and duplicate shared events in derived views
Set-valued state	Store all distinct leading-object states on the event	Distinct states without duplication	Loses the state-to-object mapping; many algorithms expect scalar values
Priority or severity aggregation	Select one state by a declared operator	Compact event-level representation	Can hide minority states and change graph counts
Coalesced transition event	Group equal transitions at the same global observation cut	Affected objects and common transition label	Must preserve order relative to every tied event

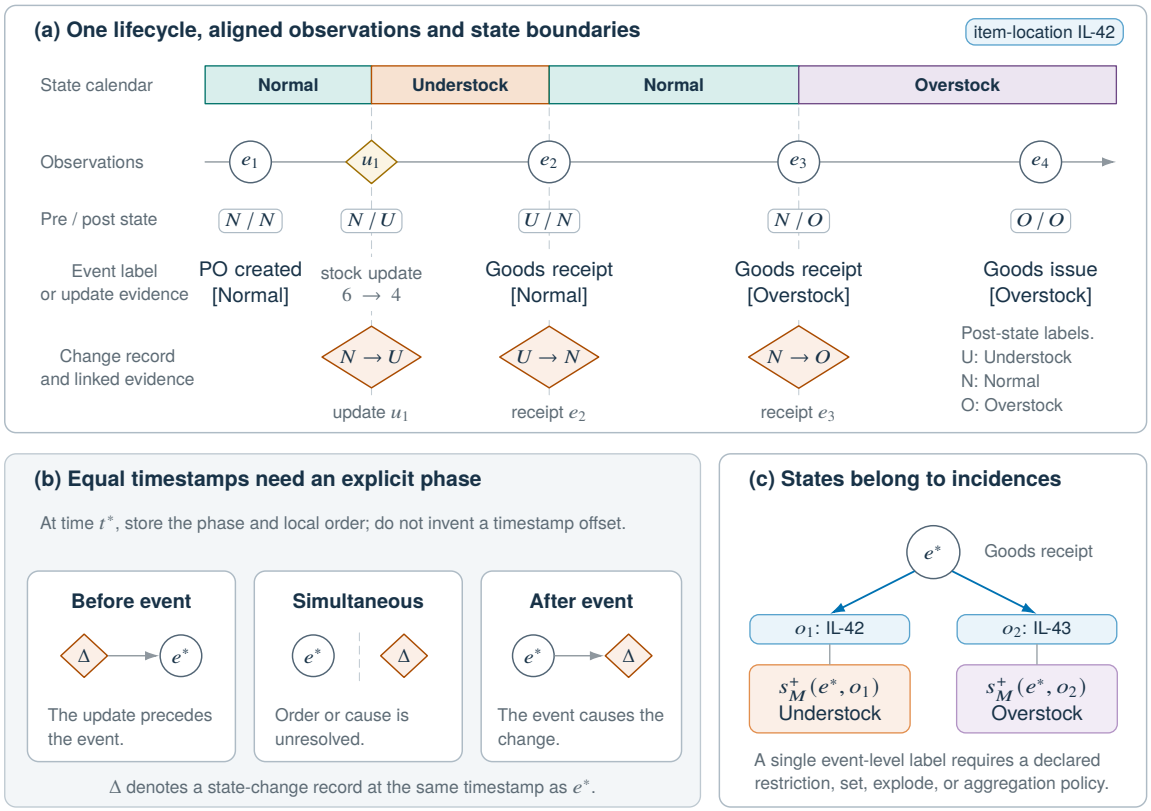

Fig. 7 State-aware enrichment and ordering semantics. A leading-object lifecycle is aligned with its state calendar, time-valid object updates, pre-state and post-state annotations, explicit change records, and state-aware labels. Equal timestamps use an explicit phase policy, and shared events retain separate annotations for each leading-object incidence.

priority or severity aggregation is compact but must be justified. A single-leading-object restriction is simple and safe when the data model guarantees at most one leading object per event.

The previous SA-OCPM construction materialized two complementary artifacts [Kretzschmann et al., 2026b]. State-aware events refine activities with state context, such as Goods Receipt[`Understock`]. State transition events make changes such as CHANGE(`Normal`, `Understock`) explicit. The formal SA-OCEL in Definition 11 generalizes this construction by retaining pre-state, post-state, cause, and incidence identity before choosing a concrete representation.

Synthetic state-change events require an ordering policy. Instead of subtracting an epsilon from timestamps, the abstract model uses the timestamp and within-timestamp observation order. An update processed before a business event places its change before that event; a change produced by post-event evaluation is placed after it. Unknown triggering information remains marked unknown, but a deterministic order is still needed for directly-follows analysis. That order is a convention, not evidence of physical precedence. If an exchange format cannot retain it, an ordering attribute or companion manifest is needed to reconstruct the same sequences.

A coalesced materialization may create one change event for several objects only when their transition label, leading type, and globally identified observation cut agree, including order relative to all tied business events. Equal timestamps and equal phase names alone are insufficient. Each object still contributes one change occurrence, and the original records remain available for object-level counts and durations.

Figure 7 depicts the enrichment of one leading-object lifecycle. The state calendar is aligned with base events and a time-valid object update; every event incidence receives pre-state and post-state annotations, and every calendar boundary produces an explicit change record with evidence. The lower panel distinguishes before-event, simultaneous or unknown, and after-event phases at one timestamp. The side panel demonstrates why two leading objects related to the same event can require different incidence states.

4.6 State-aware object-centric directly-follows graphs

A state-aware graph requires a label policy. Let h determine whether an activity is labeled by pre-state, post-state, or the state transition across the event. Common choices are

$$\lambda_{\text{pre}}(e, o) = \big(\text{act}(e), s_M^-(e, o)\big),$$

$$\lambda_{\text{post}}(e, o) = \big(\text{act}(e), s^{+}_{M}(e, o)\big),$$

$$\lambda_{\text{trans}}(e, o) = \big(\text{act}(e), s^{-}_{M}(e, o), s^{+}_{M}(e, o)\big).$$

Post-state labels are useful when an event is interpreted under the condition that is effective after its recorded updates. Pre-state labels are useful for decision points where the action is chosen from information available before execution. Transition-aware labels preserve whether the event changes the state but can produce a larger node set. The choice is part of π_M.

Definition 12 (State-aware activity label). Given a state-label policy $h \in \{\text{pre}, \text{post}, \text{trans}\}$, the state-aware label of incidence (e, o) is $\lambda^{h}_{M}(e, o)$, where λ^{h}_{M} denotes the corresponding labeling function λ_{pre}, λ_{post}, or λ_{trans} defined above. Two incidences of the same event may have different labels if their leading objects have different states.

For object o, build a state-aware sequence by replacing each lifecycle event with its incidence label and inserting explicit $\mathsf{CHANGE}(s, s')$ nodes at boundaries declared by Δ_M. Add $\mathsf{START}_{\tau,s_0}$ and END_{τ,s_f}, where s_0 and s_f are the initial and final observed states under the chosen horizon.

Definition 13 (State-aware lifecycle sequence). Let $\text{life}_L(o) = \langle e_1, \ldots, e_n \rangle$. The sequence $\text{salife}^{h}_{M}(o)$ is the ordered sequence obtained from

$$\langle \mathsf{START}_{\tau,s_0}, \lambda^{h}_{M}(e_1, o), \ldots, \lambda^{h}_{M}(e_n, o), \mathsf{END}_{\tau,s_f} \rangle$$

by inserting one $\mathsf{CHANGE}(s, s')$ occurrence for every state-change record of o, positioned according to the observation order. All changes lie between START and END, including changes for an object without events. A view using λ_{trans} may suppress a separate marker only when that exact change is represented by the incidence's pre-state and post-state; update-only changes and intermediate changes remain explicit. This choice is recorded in π_M.

Definition 14 (State-aware OCDFG). The SA-OCDFG induced by L^{SA}_{M} and label policy h is

$$G^{\text{SA}}_{M} = (N_M, E_M, w_M),$$

where N_M contains exactly the labels occurring in the leading-object sequences. For every sequence $\text{salife}^{h}_{M}(o)$, directly adjacent labels x, y contribute one occurrence to typed edge (x, y, τ); E_M contains exactly the edges with positive counts, and w_M records those counts. Thus each leading incidence occurs once, even when its event is shared or has several qualifiers. The graph can be extended with ordinary nodes and typed edges for non-leading objects, as defined by G_L. Such context edges are counted on their own object lifecycles and do not duplicate leading incidences.

The direct context extension keeps ordinary context-activity nodes separate from state-aware leading-activity nodes. Propagating leading states to non-leading lifecycles is a further view that requires an explicit mapping from shared leading incidences, including an ambiguity policy when several leading objects disagree. It is not determined by Definition 14 alone.

Proposition 1 (Recovery from state-aware lifecycle sequences). Let $\mathcal{E}$ erase state components from activity, START, and END labels and delete CHANGE occurrences from a sequence. If the state-aware construction preserves each leading incidence once and in lifecycle order, then, for every $o \in O_\tau$,

$$\mathcal{E}\big(\text{salife}^{h}_{M}(o)\big) = \text{aug}_{\tau}(o).$$

Accumulating typed adjacencies *after* this erasure therefore recovers the leading-type OCDFG of L, with its original weights.

Proof. Deleting the inserted changes leaves one labeled occurrence for each e_i, in the original order, between START and END. Erasing its state returns $\text{act}(e_i)$. This also holds for empty lifecycles, leaving START followed by END. Each object's adjacent-pair counts consequently equal those in Definition 3; summing over O_τ gives the stated weights. □

The aggregated graph G^{SA}_{M} alone is generally insufficient for this recovery. For example, two sequences $\langle a, C, b \rangle$, $\langle d, C, e \rangle$ and the alternative pair $\langle a, C, e \rangle$, $\langle d, C, b \rangle$ have the same aggregated edges through a shared change label C, but different adjacencies after its deletion. The same START and END edges can be added to both pairs. Recovering the base graph thus requires sequences or equivalent occurrence-level provenance.

Merely contracting aggregated CHANGE nodes can introduce false paths. The result preserves directly-follows counts; it does not reconstruct complete executions from a DFG.

Proposition 2 (State refinement). On the common domain $I_\tau(L)$, the partition induced by a fixed state-aware activity labeling λ^h_M refines the partition induced by $(e, o) \mapsto \mathrm{act}(e)$. If each incidence is retained once, merging state-specific activity occurrence counts recovers the leading-type activity incidence counts.

Proof. Equal state-aware labels have equal activity components. Each state-aware equivalence class is therefore contained in one activity class. These subclasses partition that class, so their cardinalities sum to its incidence count. □

Counts here refer to event-object incidences, not distinct global events: an event shared by two leading objects contributes two incidences. State refinement splits a node such as Goods Receipt into copies for `Understock`, `Normal`, and `Overstock`, exposing different paths and performance distributions. It also increases graph size, motivating filters by state, activity, object type, or transition.

Figure 8 compares an OCDFG and an SA-OCDFG for the same small log. The ordinary Goods Receipt node is refined into state-specific copies, explicit CHANGE nodes expose state boundaries, and line styles retain typed lifecycle edges for the leading and contextual object types. In this example each contextual event has one leading incidence, whose label is propagated to the contextual edge. The lower strip illustrates the erasure and aggregation operation of Proposition 1.

4.7 State episodes, state-determined segments, and behavioral patterns

Global SA-OCDFGs can still become large, especially for long-lived objects. Calendar episodes induce smaller analysis units with direct operational meaning. Fix an event-assignment policy $h \in \{\mathrm{pre}, \mathrm{post}\}$: an event belongs to the episode containing its query point p^h_i. Transition labels remain available for graph analysis, but assigning an event to one episode requires selecting one side of the transition.

Definition 15 (State episode). For $\mathrm{cal}_M(o) = \langle (b_1, s_1), \ldots, (b_r, s_r) \rangle$ and terminal cut b_{r+1}, episode $\mathrm{ep}_j(o)$ is the maximal constant-state block $[b_j, b_{j+1})$ on the ordered observation axis, labeled s_j. Its event sequence $B^h_j(o)$ contains, in lifecycle order, precisely the events whose query point p^h_i lies in this block. Its event support is $|B^h_j(o)|$, and its observed duration is $\mathrm{ts}(b_{j+1}) - \mathrm{ts}(b_j)$. An episode can have no events or zero duration;

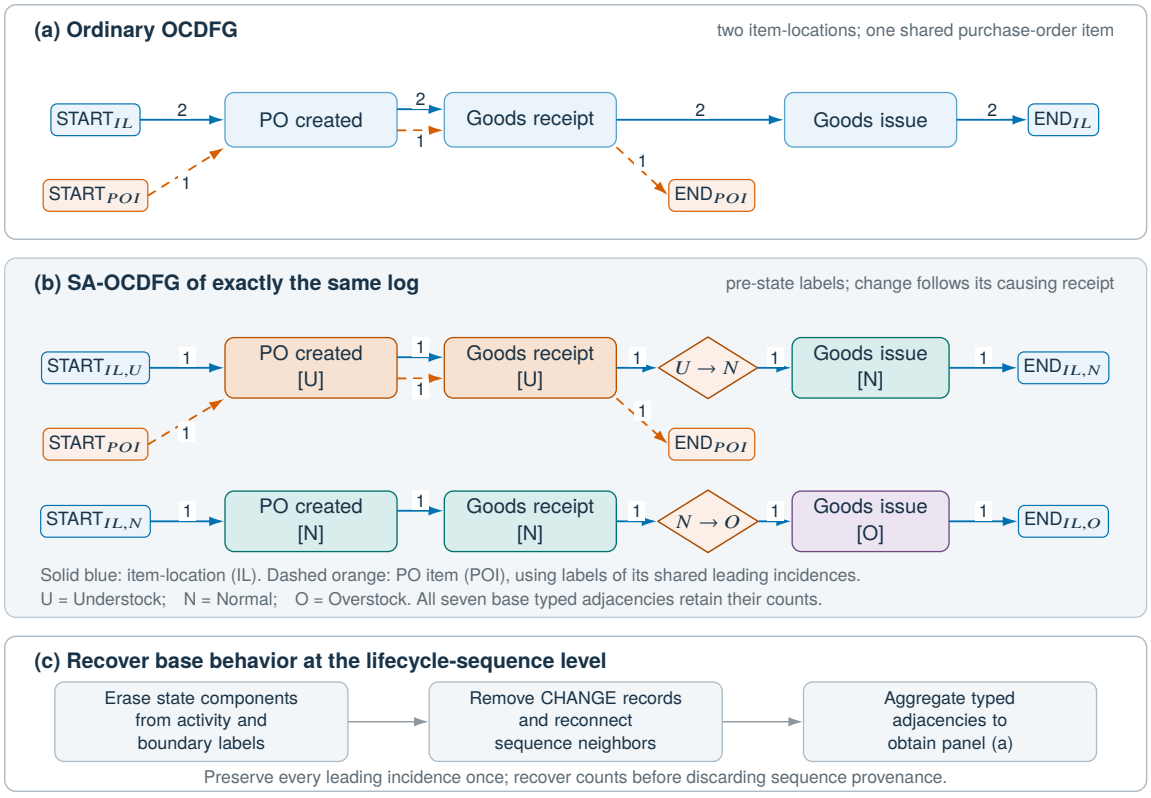

Fig. 8 From OCDFG to SA-OCDFG. State-specific activity copies and explicit CHANGE nodes refine the ordinary graph while typed object-lifecycle edges remain visible. Erasing state components and change nodes in each lifecycle sequence, then aggregating typed edges, recovers the ordinary graph.

neither condition removes its calendar boundaries.

Maximal runs of equal event labels coincide with nonempty calendar episodes only when no intervening calendar change is missed. For example, events labeled `Normal` before and after an update-only `Understock` interval belong to two different `Normal` episodes. Merging them into one event-state run would hide the shortage and misstate uninterrupted dwell. Calendar-based analyses retain the intervening episode even when it contains no events.

Definition 16 (State-determined segments). An intra-state segment consists of the state-aware labels of $B^h_j(o)$, bounded by $\mathsf{START}_{\tau,s_j}$ and END_{τ,s_j}. An inter-state segment uses two adjacent calendar episodes $j, j+1$. It concatenates START, a selected suffix of $B^h_j(o)$, the boundary marker $\mathsf{CHANGE}(s_j, s_{j+1})$, a selected prefix of $B^h_{j+1}(o)$, and END, with boundary states s_j, s_{j+1}. Selection retains either both complete event sequences or at most a fixed positive number of events on each side. Empty event sequences are permitted and are never skipped to manufacture a transition between nonadjacent episodes.

The complete-episode policy preserves all assigned events in the two episodes. A bounded radius focuses on the boundary. Segment markers separate event assignments to episodes; this analytical ordering need not equal the trigger order of Definition 13. In particular, a post-state event can be assigned to the target episode even when its update produces the boundary after execution. Segment order must therefore not be interpreted as causal precedence. The assignment, radius, and any exclusion of empty segments are stored in the pattern configuration, and support is reported relative to the retained segments.

Each segment is transformed into an aggregated local graph. There is one control-flow node per distinct state-aware activity or auxiliary label and one separately tagged object node per participating object type. Let O_z be the union of objects related to the selected original events. Directly-follows edges summarize consecutive labels of the segment sequence and carry its leading type. Event-to-object-type edges summarize event participation. Object-to-object-type edges summarize the declared visible relations whose two endpoints belong to O_z.

Definition 17 (Segment graph). For segment z, its local graph is

$$H_z = \left(V^C_z \cup V^O_z,\ E^{DF}_z \cup E^{EO}_z \cup E^{OO}_z,\ \mathrm{lab}_z,\ w_z\right),$$

where V^C_z and V^O_z are disjoint node sets with the labels just specified. Edges are keyed by their kind, ordered endpoints, and retained type or qualifier labels. The directly-follows weight counts adjacent positions in the segment. The participation weight counts selected event-object pairs, or qualified triples when qualifiers are retained. The object-relation weight counts distinct ordered pairs in O_z, or qualified triples under the qualifier policy, once per segment. Auxiliary markers have no invented event-object relations. Edges with the same key are summed, and only positive-weight edges are retained. The function lab_z stores these labels and w_z stores the weights.

The graph retains the selected state boundary and type-level context while discarding instance identities. It does not preserve which individual objects share events or the complete activity sequence. Distinct sequences can therefore have the same local graph.

Definition 18 (Structural signature and behavioral pattern). The structural signature $\mathrm{sig}(H_z)$ is the labeled graph obtained from H_z after removing occurrence weights and instance identifiers while retaining node labels, edge labels, edge direction, object types, state labels, and configured qualifiers. Two segments are structurally equivalent if their signatures are isomorphic under a label-preserving graph isomorphism. A state-aware behavioral pattern is an equivalence class of structurally equivalent segments.

Segment instances retain distinct identifiers even when their graphs are equal. For a pattern P, define

$$\mathrm{supp}(P) = |P|, \qquad \mathrm{mass}(P) = \sum_{z \in P} \sum_{a \in E^{DF}_z} w_z(a).$$

Support counts segments, not distinct objects or independent observations; overlapping inter-state segments can share events. Control-flow mass includes the declared auxiliary edges. Patterns are ranked by descending support and then mass, with a canonical signature used to break remaining ties. Additional criteria can emphasize duration, object coverage, severity, or statistical contrast. Since node labels are unique within each tagged node kind, a sorted list of labeled nodes and edge keys is an exact structural signature for this graph definition. Removing weights deliberately ignores repetition counts, so a pattern is an equivalence class of aggregated structures rather than an exact repeated execution.

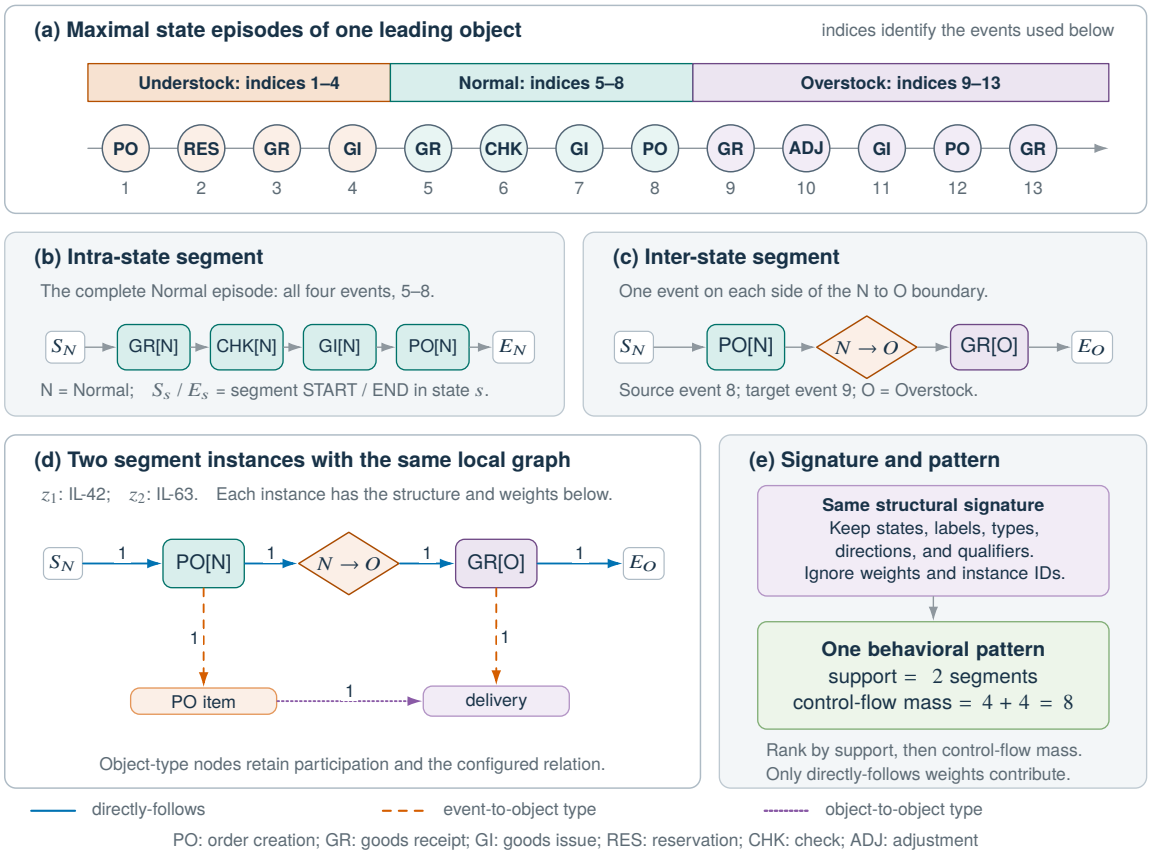

Fig. 9 From state episodes to recurring behavioral patterns. Maximal state episodes induce intra-state and inter-state segments; local typed graphs aggregate control flow and object context (the leading-type node is omitted from the illustrated graph); structural signatures remove instance-specific identifiers and weights; and equivalent signatures aggregate into ranked patterns.

Intra-state patterns answer questions such as which routines repeatedly occur while an item remains overstocked. Inter-state patterns answer questions such as which local behavior accompanies a transition from `Normal` to `Understock` or from `Down` to `Recovery`. Because the local graph contains object types, a pattern can reveal whether a purchase order, sales order, supplier, work order, or component is present even when the activity sequence is similar.

Figure 9 shows one leading lifecycle segmented into three maximal state episodes. One intra-state segment and one bounded inter-state segment are extracted, segment instances are converted into local typed graphs, and instance identifiers and weights are removed to form a structural signature. Equivalent signatures are then aggregated into a ranked behavioral pattern with support and control-flow mass.

4.8 State-aware analyses and their advantages

The state layer supports a family of analyses rather than one algorithm. Figure 10 organizes these analyses as an evidence-maturity landscape. State construction, assignment, validation, and provenance form the foundation; descriptive and diagnostic layers contain temporal indicators, state-conditioned models, conformance, patterns, boundary conditions, and feature associations; and the prospective layer contains prediction, decision support, and intervention evaluation. Table 5 summarizes the main analysis families, their inputs, outputs, and advantages.

Which results require the state layer? The outputs in Table 5 have explicit dependencies. Occupancy and recovery require a state calendar; state-conditioned control flow and conformance require the state of each relevant event-object incidence; intra-state and inter-state patterns require state-derived segment boundaries; and state-exposure models and undesirable-transition prediction require state-defined outcomes. Removing these constructions leaves activity frequencies, execution structures, and raw measurements, but does not leave the same analytical questions answerable. For example, two histories with identical activities and object links can have different stock trajectories and therefore different shortage durations. A summary that omits those conditions cannot recover the difference. SA-OCPM provides a common construction of the missing semantics

Table 5: Catalog of state-aware analysis families.

Family	Question	Main output	Advantage of state awareness	Tool
Occupancy and dwell	How long are objects in each condition?	Exposure, episode duration, censoring-aware summaries	Measures the operational condition rather than event density	Partial
Transitions and recovery	How do objects enter, leave, and return?	Counts, rates, spectra, recovery, recurrence, stuck objects	Separates entry, persistence, and restoration	Partial
State-conditioned control flow	What behavior occurs under each condition?	SA-OCDFG, state-specific paths and times	Splits identical activities by operational meaning	Yes
Conformance	Are state transitions and responses allowed and timely?	Violations, missing actions, deadlines, relational checks	Localizes policy to the condition in which it applies	Partial
Patterns	Which local behaviors recur inside or across states?	Ranked intra-state and inter-state graph signatures	Distinguishes maintenance from transition behavior	Yes
Boundary conditions	What characterizes entry and exit of learned states?	Feature contrasts, entry or exit DFGs, attribute overlays	Makes unsupervised states inspectable	Yes
Association and rules	Which features accompany exposure or transition?	Correlations, contrasts, rules, cohort differences	Provides state-derived outcomes and stratification	Partial
Structural or causal models	How might operational factors relate to state outcomes?	DAG or SEM parameters and sensitivity analyses	Uses interpretable time-in-state outcomes	Explor-atory
Prediction	Which object will enter or leave a state, and when?	Risk, next state, lead time, time to recovery	Creates operationally meaningful targets	Not yet end-to-end
Decision support	Which action is appropriate for the current condition?	Evidence-linked candidate action and policy matrix	Aligns recommendation with state and context	Partial

instead of requiring a separate reconstruction for every task. This representational requirement does not imply that state features always predict better than raw features: a raw-feature predictor can compete on a state-defined target once that target has been constructed.

4.8.1 State occupancy, persistence, and transition performance

The state calendar makes time in state a first-class measure. On the horizon $[T_o^0, T_o^1)$, let $\sigma_M(o,t)$ be the time-state defined by the calendar, and let μ denote duration (Lebesgue measure on the common time scale). For state s, define exposure by

$$D_o(s) = \mu\big(\{t \in [T_o^0, T_o^1) \mid \sigma_M(o,t) = s\}\big).$$

The normalized exposure is $d_o(s) = D_o(s)/(T_o^1 - T_o^0)$. Because the calendar partitions the horizon,

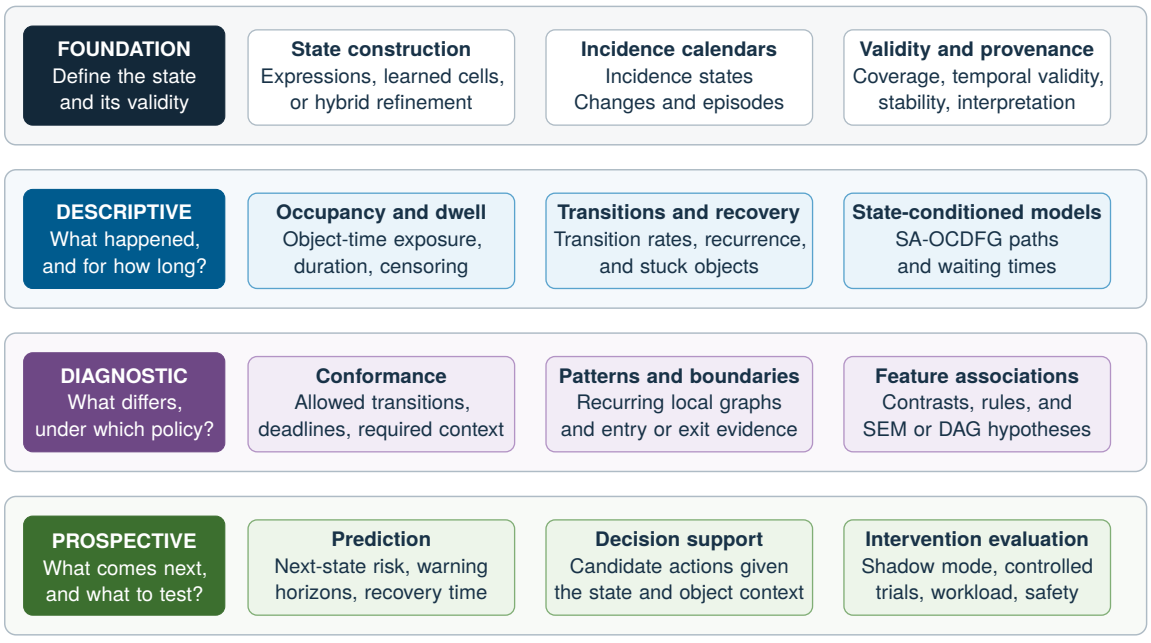

Fig. 10 Landscape of state-aware analyses. Construction, assignment, validation, and provenance support descriptive occupancy, transition, and control-flow views; diagnostic conformance, pattern, boundary, and association analyses; and prospective prediction, decision-support, and intervention-evaluation studies.

$\sum_{s \in S_\tau} D_o(s) = T_o^1 - T_o^0$ and $\sum_s d_o(s) = 1$, including `Unknown`. Aggregate exposure $D(s) = \sum_{o \in O_\tau} D_o(s)$ measures total object-time. This differs from counting state-labeled events: a state with few events can dominate elapsed time.

Censoring must be explicit. The initial episode can have begun before the horizon, and the final episode can continue beyond it. Report their observed durations with the appropriate boundary flags. Summaries restricted to episodes with observed entry and exit describe that selected population and can be biased toward short episodes. Survival methods require their own censoring assumptions. Zero-duration episodes contribute transitions and event support but no exposure.

For each object o, let $\mathrm{cal}_M(o)$ contain the consecutive states $s_1^o, \ldots, s_{r_o}^o$. The transition count is

$$N(s, s') = \left| \{(o, j) \mid 1 \le j < r_o \wedge s_j^o = s \wedge s_{j+1}^o = s'\} \right|.$$

A conditional transition proportion is $\widehat{P}(s' \mid s) = N(s, s') / \sum_{u \in S_\tau} N(s, u)$ when the denominator is nonzero. It describes the destination conditional on an observed exit from s; it is not a fixed-horizon transition probability and requires no Markov assumption. In particular, $N(s, s) = 0$, and $\sum_{s,s'} N(s, s') = \sum_{o \in O_\tau} (r_o - 1) = |\Delta_M|$. Zero-exit rows are undefined rather than normalized to zero. Stratification can reveal differences across suppliers, periods, or policy regimes.

Dwell time is the duration of an episode. For each state s, the dwell-time multiset supports minimum, median, mean, maximum, quantiles, and survival curves. Long dwell in an undesirable state identifies persistent problems. Short alternating dwell can indicate oscillation, measurement noise, or an unstable policy. A minimum-duration or hysteresis rule can reduce meaningless chattering when state boundaries are threshold based.

Recovery analysis requires disjoint undesirable and acceptable sets $U, G \subseteq S_\tau$. An entry into U is a calendar boundary from outside U to inside it; changes between members of U do not start another recovery clock. Fix a minimum acceptable duration $\eta \ge 0$. The recovery boundary is the first later boundary, in observation order, that enters G and is followed by at least η observed time continuously in G. Internal transitions within G are allowed. Recovery duration is the difference of boundary timestamps and can be zero for ordered changes at the same time. An acceptable final interval shorter than η does not establish stable recovery. When recovery is timed at entry into G, right-censor at the last candidate entry time with a complete η-long follow-up, namely $T_o^1 - \eta$, rather than automatically at T_o^1. Entries into U after that cutoff have insufficient follow-up for this analysis. Objects already in U at the horizon start have unknown entry time and must be reported separately. `Unknown` does not count as acceptable unless explicitly justified.

Recurrence measures how often an object returns to an undesirable state after recovery. A stuck-object view identifies objects whose final observed state is undesirable and whose current episode exceeds a threshold. These indicators provide operational queues for investigation. Their advantage over generic throughput time is that they measure the condition the organization actually seeks to avoid or restore.

A time perspective aggregates state frequency or object-time over calendar buckets. It can reveal seasonality, changes after policy deployment, synchronized transitions, and bursts of degradation. A transition-performance spectrum plots individual source-to-target transitions by time and duration. One-way spectra measure the time from entry into a source state to first arrival at a target. Roundtrip spectra measure source to target and back to source. The distinction prevents a fast temporary recovery from being confused with durable stabilization.

4.8.2 State-conditioned control flow and conformance

An SA-OCDFG separates the behavior observed under different states. Frequency filtering can answer which activities and object-type interactions dominate `Understock`, `Normal`, and `Overstock`. Performance annotations can compare waiting times on the same directly-follows relation across states. For example, Purchase Order Created followed by Goods Receipt may have a much longer waiting-time distribution in `Understock` than in `Normal`, even when the ordinary OCDFG merges both occurrences.

State-conditioned models also support conformance. Let $C_S = (S_\tau, T_{\text{allow}})$ with $T_{\text{allow}} \subseteq S_\tau \times S_\tau$ be an allowed state-transition graph. A state transition δ from s to s' violates the state policy if $(s, s') \notin T_{\text{allow}}$. A richer policy can be a relation

$$\mathit{Allow} \subseteq S_\tau \times A \times S_\tau,$$

where (s,a,s') states that activity a is permitted when moving from s to s'. Another relation can specify required actions after entry into a state, deadlines, or object-context conditions. For example, entry into `Critical Understock` may require an escalation event within four hours and an open replenishment object.

State-aware conformance can be assessed at several levels. Transition conformance checks the sequence of states. State-conditioned control-flow conformance checks activities under each state. Temporal conformance checks maximum dwell or response times. Relational conformance checks the presence or role of related objects.

Model-based techniques such as alignments can be extended with state labels, while graph-based diagnostics can compare observed SA-OCDFG edges with allowed edges.

The main advantage is semantic localization. An activity may be acceptable in `Normal` but prohibited in `Blocked`. A delay may be harmless in a stable state but critical after entry into `At Risk`. A generic conformance score can hide these distinctions. State-aware diagnostics can state which policy was violated, for which leading object, during which episode, and with which supporting objects.

Conformance results must still account for state-definition error. A violation caused by a missing attribute or an unstable learned state should not be treated as equivalent to a confirmed operational deviation. Confidence and `Unknown` assignments can be propagated into the report, and low-confidence episodes can be placed in a review queue.

4.8.3 Patterns and boundary conditions

State-aware behavioral patterns summarize recurring local graphs inside states and around changes. Intra-state pattern support reveals common routines that maintain a condition. Inter-state pattern support reveals common routes of entry and exit. Comparing pattern distributions across object cohorts can identify context-specific mechanisms. A pattern containing repeated Sales Order Created and Goods Issue while an item remains `Understock` suggests demand pressure. A `Normal` to `Overstock` pattern containing a large Goods Receipt and no nearby demand event suggests replenishment overshoot.

Pattern ranking by support prioritizes common behavior, but frequent does not necessarily mean important. Severity-weighted support can multiply each instance by duration in the resulting undesirable state, estimated cost, service impact, or another outcome. To compare behavior across states, contrastive support uses a separately declared state-erased projection of the signature; the state-preserving signatures of Definition 18 cannot themselves match across different state labels. Statistical testing or confidence intervals should be used when interpreting small differences, and multiple-comparison control is needed when many patterns are screened.

Automatic states require explanation. For a target state g, define entry windows E_g as windows whose previous assignment is not g and current assignment is g. Define stable windows S_g as windows whose previous and current assignments are both g. Feature contrasts compare E_g with S_g or with entry windows of neighboring states. For numeric feature j, let x_j denote its value in a window. A standardized contrast can be written as

$$\Delta_j(g) = \frac{\text{mean}_{E_g}(x_j) - \text{mean}_{S_g}(x_j)}{\text{pooled sd}_j},$$

where, writing n_E, n_S for group sizes and $v_{E,j}, v_{S,j}$ for within-group sample variances,

$$\text{pooled sd}_j = \sqrt{\frac{(n_E - 1)v_{E,j} + (n_S - 1)v_{S,j}}{n_E + n_S - 2}}.$$

The contrast is reported only when both group sizes are at least two and this denominator is positive. Initial windows without a previous assignment are excluded from the entry/stable comparison. Overlapping windows are dependent, so the descriptive contrast alone does not supply an independent-sample significance test.

Categorical features can be compared through prevalence differences or odds ratios. Activity and object-type frequencies can be displayed as boundary DFGs. Exit windows are analyzed analogously. These contrasts do not prove that a feature causes the transition, but they make the learned boundary inspectable.

A useful validation is agreement between learned and specified states. If an automatic cell is colored mostly by `Understock` under a manual stock definition, its interpretation is supported. If one cell mixes all stock regimes but has a distinct interaction pattern, it may capture a different operational dimension. Agreement measures such as adjusted Rand index, normalized mutual information, or best-matching purity can be reported, but low agreement is not automatically failure because the two state notions may answer different questions.

4.8.4 Feature association, rule mining, and structural analysis

State-aware outcomes can be connected to object-level features. For each leading object, construct a feature vector with activity counts, transition counts, object-interaction counts, durations, supplier or product characteristics, and state exposures. Correlation analysis can compare the distribution of a feature inside and outside a dominant state or relate the feature to $d_o(s)$. Rule mining can identify combinations associated with prolonged states or repeated transitions, extending object-centric root-cause approaches [Knopp et al., 2025].

Association analysis should respect dependence. Multiple episodes from one object are not independent, and objects in the same location, supplier, or production line may share unobserved conditions. Clustered standard errors, multilevel models, or object-level aggregation can reduce misleading certainty. Temporal direction should also be enforced when the goal is explanation: features measured after a transition cannot explain the transition without a retrospective interpretation.

Structural equation modeling offers one way to encode a theory of how latent operational factors relate to state exposure [Berti et al., 2026a, Qafari and van der Aalst, 2020]. In inventory management, observable indicators may load onto Demand, Supply, Batchiness, and Buffering constructs, which are then linked to `Understock` and `Overstock` exposure. The state-aware log contributes well-defined temporal outcomes and process-derived indicators.

The state abstraction can improve such models by reducing outcome heterogeneity. Instead of using one generic delay or cost value, an analysis can model time in `Understock` and time in `Overstock` separately. However, a state label does not create causal identification. A valid causal interpretation requires a defensible directed acyclic graph, measurement assumptions, temporal precedence, treatment and outcome definitions, sufficient adjustment, and sensitivity to unmeasured confounding. FLOWVAULT therefore presents its causal workbench as exploratory structural analysis unless those assumptions are established externally.

4.8.5 Prediction and process-aware decision support

A state calendar creates targets for predictive monitoring. Models can estimate the next state, the probability of entering an undesirable state within a horizon, the time to recovery, or the risk of recurrence. Inputs can include recent event-object windows, current state, dwell time, related-object features, and process-model conformance. State-aware prediction is often easier to interpret than prediction of a distant final outcome because the target corresponds to an operational regime.

Evaluation must use temporal splits and must prevent leakage from future object attributes. Calibration, precision-recall, time-dependent concordance, lead time, and false-alert burden are relevant metrics. Prediction accuracy alone is insufficient. A warning that arrives after the state transition or that triggers too many alerts may have little operational value.

Decision support maps evidence to candidate actions. A policy matrix can associate state, boundary condition, and object context with actions such as expedite replenishment, review order quantity, schedule inspection, initiate maintenance, or escalate a block. The action should be presented with the evidence that supports it: current state, duration, transition history, pattern, relevant objects, and uncertainty.

Prescriptive claims require stronger evaluation than descriptive findings. A historical association between an action and recovery can reflect selection bias because difficult cases receive different actions. Candidate interventions should be tested through controlled trials, phased rollout, quasi-experimental designs, or at least prospective shadow-mode evaluation. Safety-critical domains should not automate actions solely from an unvalidated learned state.

4.8.6 Comparative and multi-state analyses

Different state notions can coexist. Analysts can compare a manual stock state with an automatic execution state, or a machine-health state with a production-mode state. For the same leading object and a common horizon, carry both notions forward on the union of their observation axes. The joint assignment is then the pair of their states in $S_1 \times S_2$, but the product can become sparse. Notions for different leading types require an explicit relation between their objects before a joint assignment is defined. Cross-tabulation, conditional transition analysis, and state alignment are usually more readable than building one large product-state graph.

Cohort comparison asks whether state exposure, transitions, patterns, or recovery differ by supplier, plant, product group, customer segment, or policy period. A state definition should remain fixed when cohorts are compared unless the purpose is explicitly to compare state notions. Otherwise, an apparent performance difference may be caused by different thresholds.

The overarching advantage of SA-OCPM is not merely additional detail. It offers a controlled way to reduce heterogeneity, align process behavior with operational objectives, separate persistence from boundary behavior, and retain the multi-object evidence needed for diagnosis. The cost is additional modeling choice. State construction, horizon, aggregation, confidence, and granularity must therefore be reported as part of the analysis.

4.9 End-to-end procedure, properties, and complexity

The complete workflow accepts an OCEL, a leading object type, a state-construction mode, a materialization policy, and an analysis plan. It returns a versioned SA-OCEL and selected state-aware outputs. The steps are summarized in Algorithm 1 and Figure 11.

Figure 11 shows the end-to-end method as a gated pipeline. Data validation and perspective selection precede state construction; a quality gate checks coverage, temporal validity, stability, and interpretation before incidence annotation. Optional materialization, SA-OCDFG discovery, segment and pattern construction, selected analyses, expert review, and versioned export follow. Failed quality or review gates return the workflow to state construction or perspective refinement.

Algorithm 1 End-to-end construction and analysis of an SA-OCPM perspective

1: Validate the OCEL schema, identifiers, timestamps, relations, and object-attribute histories.
Validated base log L and checksum
2: Select leading object type τ, objective, observation horizon, and tie or phase policy.
Documented analytical perspective
3: Construct M_τ through an expression, an automatic model, or a hybrid refinement.
State set, fitted or rule definition, provenance ρ_τ
4: Evaluate coverage, `Unknown` exposure, temporal admissibility, stability, and interpretability.
Accept, revise, or mark exploratory
5: Annotate every leading incidence with pre-state, post-state, evidence, and optional confidence.
Annotation relation ann_M
6: Create state-change records and choose an event-level materialization only when needed.
Δ_M, π_M, and SA-OCEL L_M^{SA}
7: Derive state-aware lifecycle sequences and the SA-OCDFG. G_M^{SA} with declared label policy
8: Create episodes, segments, local graphs, signatures, and ranked patterns as requested.
Intra-state and inter-state outputs
9: Compute temporal, conformance, diagnostic, association, prediction, or decision analyses.
Evidence tied to objects, states, and context
10: Review findings, perform sensitivity checks, and export versioned configurations and outputs.
Reproducible analysis package and refinement loop

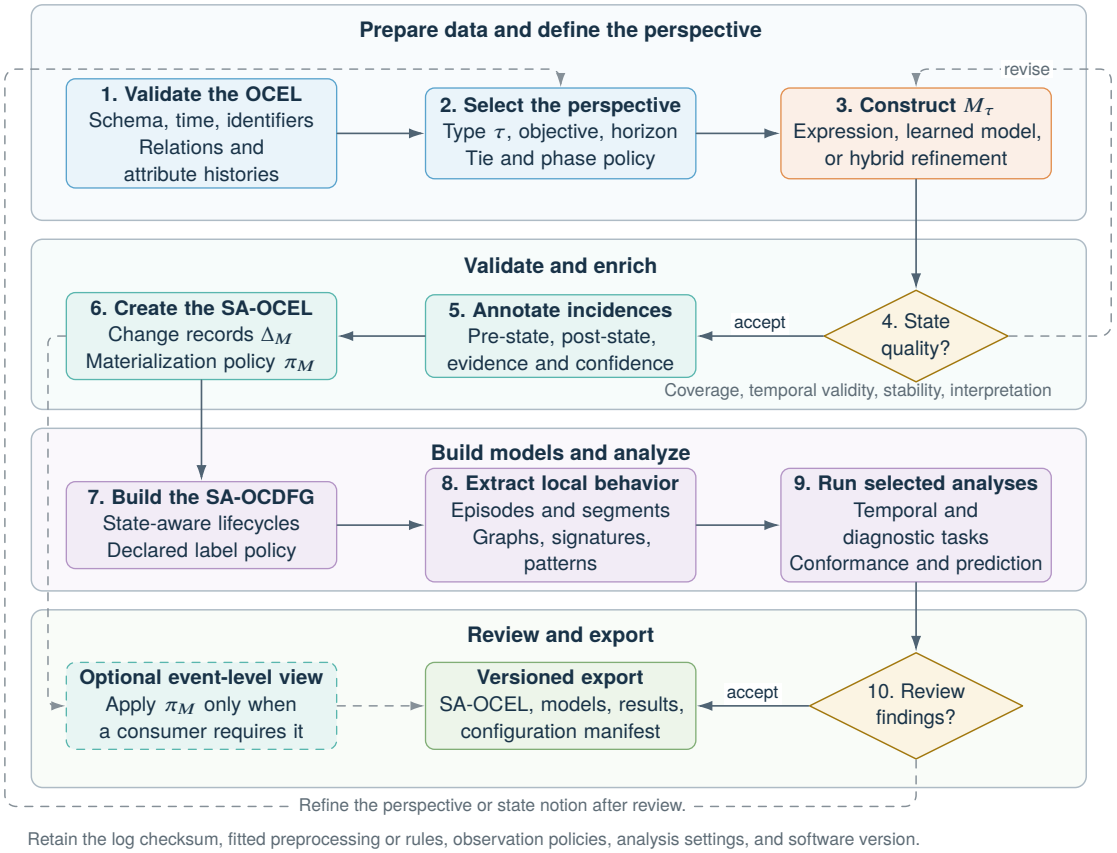

Fig. 11 End-to-end SA-OCPM workflow. Quality and expert-review gates separate exploratory state construction from accepted incidence annotation, graph and pattern discovery, analysis, and versioned export. Dashed paths denote optional materialization and refinement loops.

The state-validation gate is essential. For an expression, checks include branch coverage, `Unknown` exposure, threshold sensitivity, and expert review of boundary cases. For an automatic model, checks include feature leakage, reconstruction or quantization quality, cell occupancy, stability across seeds or samples, temporal consistency, and interpretability. A state notion that fails these checks can still be used exploratorily, but it should not be used for conformance or intervention claims without qualification.

The conservative design supports reproducibility. The base OCEL is identified by a checksum. The leading type, state expression or learned preprocessing, model parameters, observation horizon, tie policy, and materialization are stored in ρ_τ and π_M. Pattern settings, graph filters, and analysis parameters are stored separately. Re-running the same configuration on the same base log should reproduce the state annotations and deterministic outputs.

The computational cost depends on the construction mode and selected analyses. Let n_P be the total number of evaluation points and m the number of expression predicates. Assignment takes $O(n_P m)$ only when context extraction and each predicate evaluation have bounded cost. Computing relational aggregates or searching attribute histories adds its own cost; an index alone does not guarantee constant-time queries. Lifecycle traversal is linear in the number of related event occurrences after sorting or using existing indexes; calendar and transition construction are linear in the number of observation points once their states are available. OCDFG and SA-OCDFG accumulation are linear in the total length of the relevant lifecycle sequences.

For W explicit concatenated windows of length w and event-feature dimension d, materialization costs $O(Wwd)$ in addition to computing the event features. PCA and SOM training depend on input dimension, component count, grid size, iterations, and implementation. Sliding windows can be updated incrementally for aggregate encodings. Pattern construction is output-sensitive: it processes the selected event occurrences and retrieved context relations. For a local graph with v nodes and e edges, sorting its uniquely labeled node and edge records gives an exact signature in $O(v \log v + e \log e)$ comparisons. Grouping uses these signatures; hashes, if used as an index, require equality checks to resolve collisions. No general graph-isomorphism procedure is needed for Definition 17.

Memory use is influenced by whether the base log, active filtered log, state annotations, windows, graphs, and pattern instances are retained simultaneously. String interning, typed compact values, lifecycle indexes, streaming feature construction, and storing pattern signatures rather than duplicated graphs can improve scalability. Section 5 describes how FLOWVAULT implements several of these choices.

The workflow also supports incremental deployment. An organization can begin with a simple expression-based state and occupancy indicators, then add transition performance, SA-OCDFGs, and patterns. Automatic state discovery can be used to challenge or enrich the manual taxonomy. Predictive or prescriptive components should be added only after state stability and analytical utility have been demonstrated.

5 Tool: FLOWVAULT

5.1 Purpose and architecture

FLOWVAULT is a browser-based OCEL 2.0 workbench that operationalizes the main SA-OCPM concepts [Berti, 2026c]. The tool is implemented as an Angular application with a Rust analysis core compiled to WebAssembly. OCEL files are parsed and analyzed locally in the browser, and no server-side analysis API is required. The core analysis path does not require uploading the log to a remote service. Optional LLM assistance sends the user prompt and selected log metadata to the configured model provider, as described below.

Figure 12 presents the FLOWVAULT architecture in four layers. Multi-format OCEL input feeds a compact Rust analysis core with typed storage, lifecycle indexes, validation, state construction, graph, pattern, feature, and export engines. Typed WebAssembly calls connect the core to the Angular workspace, whose pages provide state editing and detection, graphs, KPIs, patterns, lifecycles, correlations, and structural analysis. Visualizations, CSV feature tables, and OCEL exports leave the same local-browser boundary.

The Rust core uses a compact representation. Repeated event identifiers, object identifiers, type names, attribute names, qualifiers, and string values are interned and referenced by integer symbols. Event timestamps and object-attribute timestamps preserve microsecond precision. Attribute values remain typed as string, timestamp, integer, float, or Boolean. Every object has a timestamp-ordered lifecycle index of related event positions. These choices reduce repeated storage and speed up lifecycle-based analyses.

FLOWVAULT keeps a source log and an active filtered view in memory. Applying an expression enriches the source log and then rebuilds the active view; filtering therefore preserves stored state attributes while changing the events and relations used by graph and pattern analyses. Applying discovered states computes them on the active view and writes the resulting event labels back to the source log. A filter change does not by itself refit these learned states. Reproduction therefore requires both the filter configuration and the population on which a state notion was evaluated, and the unmodified input file should be retained separately.

5.2 Data import, validation, filtering, and export

The entry page offers two ways to start an analysis (Figure 13). Users can import their own OCEL 2.0 file through drag-and-drop or file selection, or try the workbench with one of the bundled example logs. The available examples cover purchase-to-pay, order management, container logistics, and inventory simulation, with JSON

and XML alternatives. This allows users to explore the analysis workflow before preparing their own data; the workbench is not restricted to these examples.

The tool imports OCEL 2.0 in JSON, XML, CSV, SQLite, and bundled Parquet formats. Gzip-compressed JSON, XML, and CSV are supported. The importer validates identifiers and types, attribute declarations, scalar values, timestamps, relationship targets, object-change consistency, CSV reconstruction rules, SQLite mappings and table schemas, bundle paths, and Parquet schemas. The active log can be exported to the supported formats, allowing state-enriched or filtered data to be transferred to another environment.

The Statistics page reports event, object, event-to-object, and object-to-object counts. When a filter is active, counts are shown relative to the original log. Activity and object-type filters can be combined through a filter chain. This provides a controlled way to focus state construction on a relevant sublog, but it also means that the filter configuration must be recorded in a reproducible analysis.

The compact data layer follows the time-varying object-attribute semantics needed by expression-based states. At an event timestamp, an object field resolves to the latest object-attribute value at or before that time. ISO 8601 timestamps with offsets are normalized to UTC on export, while timestamps without offsets are treated as UTC by the current importer. Analysts should confirm that this convention matches the source system.

5.3 Expression-based state enrichment

FLOWVAULT provides a SQL-like state editor. A typical query has the following shape:

Listing 1: Example expression-based state definition in FLOWVAULT

```
STATE state FOR LEADING OBJECT TYPE 'Order' AS CASE
WHEN object.status IS NOT NULL THEN object.status
WHEN object.is_blocked = 'Yes' THEN 'Blocked'
WHEN event.type LIKE '%cancel%' THEN 'Exception'
ELSE 'Normal'
END
```

The query can reference event.id, event.type, event.time, event attributes, object.id, object.type, and object attributes of related leading objects. It supports equality and order comparisons, LIKE with percent wildcards, IS NULL, IS NOT NULL, AND, OR, NOT, and parentheses. Results can be literals or field references. Reapplying a query replaces the derived event attribute named state.

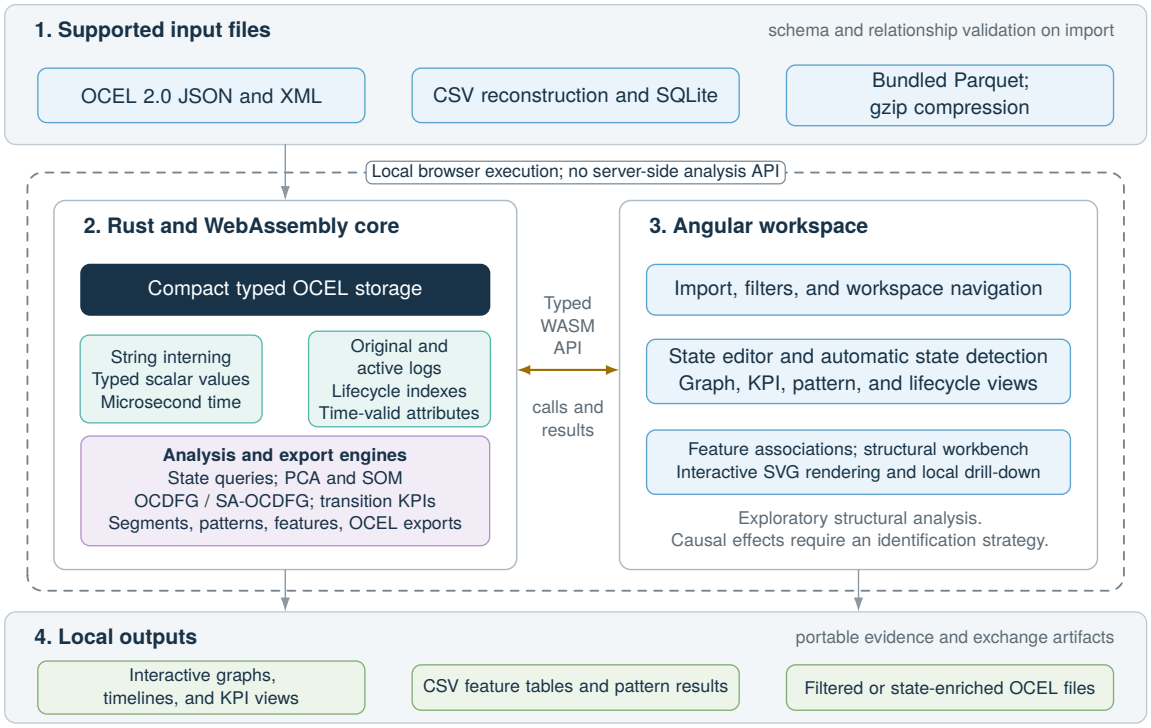

Fig. 12 FLOWVAULT architecture and analysis workspace. Supported OCEL files enter a compact Rust and WebAssembly analysis core, the Angular workspace exposes state and process-analysis pages, and visualizations, feature tables, and enriched OCELs are produced locally in the browser. The optional external LLM authoring call is outside this core analysis path.

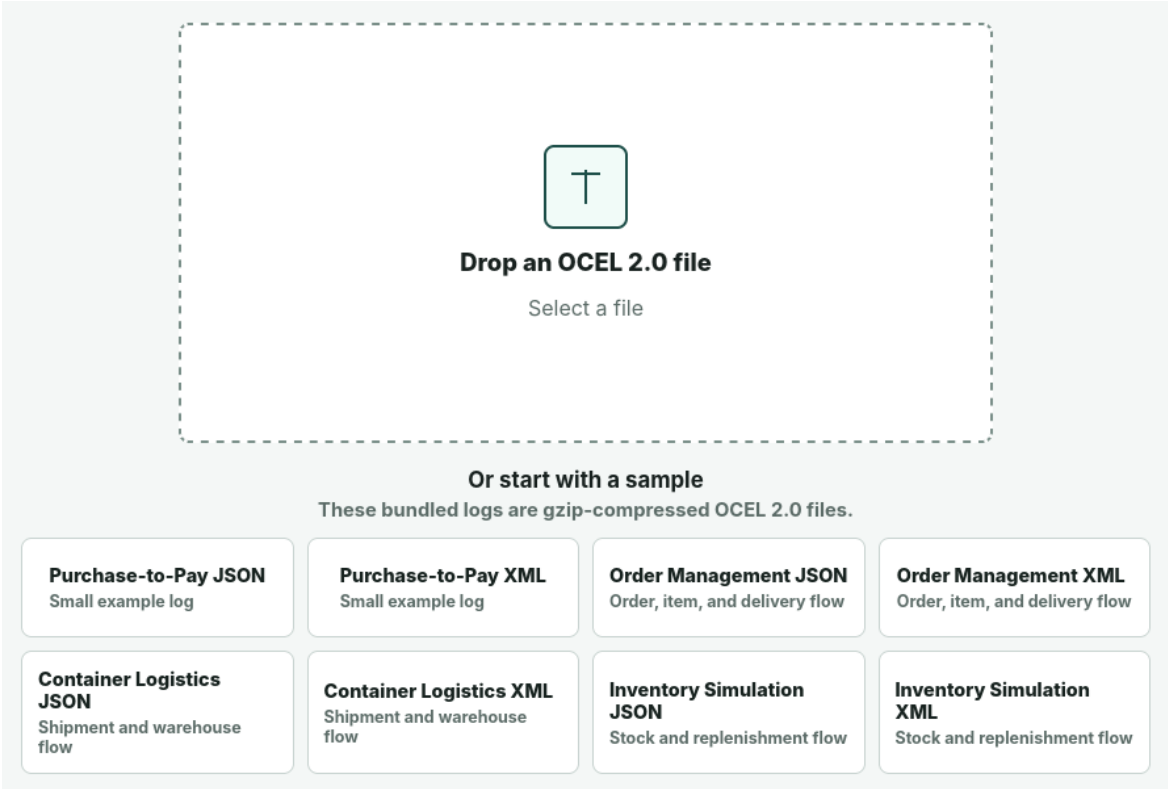

Fig. 13 FLOWVAULT entry page. Users can import an OCEL 2.0 file or select a bundled sample log to try the application. The screenshot retains the import area and all example-log choices; empty outer margins are omitted for legibility.

The current implementation materializes state at event level. Only events related to at least one object of the selected leading type receive a state. An object-based condition is true when at least one related leading object satisfies the complete condition. This existential aggregation is a practical policy, but it can merge different incidence states when an event relates to several leading objects. Table 6 maps the formal framework to the current tool and identifies this difference explicitly.

The editor offers named presets for bundled example logs, including stock status and stock movement for an inventory-management log, fulfillment and exception views for order management, and shipment or load-planning views for container logistics. Presets are starting points. Their thresholds and semantics should be reviewed before analytical use.

State definitions can also be authored with large language model (LLM) assistance. In the *Add Event State* dialog, the analyst selects *Ask LLM*, enters a natural-language request, and invokes *Generate Expression* (Figure 14). The LLM proposes a SQL-like state definition for the log, including a leading object type and conditional state assignments. The illustrated request produces an expression for the `packages` object type, with branches such as `Delivery Failure` and `Delivered`. The proposed expression is visible for inspection and editing before application. Once accepted, it is evaluated through the same expression-based enrichment mechanism as a handwritten query. LLM assistance therefore supports the authoring of a state notion; the accepted expression specifies the subsequent assignments. The request sends the prompt, filename, event and object counts, type names, and up to 30 available feature-column names to the configured provider. Core log processing remains local, but this optional authoring operation communicates externally. The quality of generated expressions is outside the scope of the experiments reported in this paper.

After enrichment, the exported OCEL contains the derived state as an ordinary typed event attribute. This makes the result interoperable with software that does not understand the abstract SA-OCEL tuple. It also means that state-notion provenance should be stored alongside the export or encoded in a companion manifest, because a plain event attribute does not capture the complete rule, leading type, or aggregation policy.

5.4 Automatic state detection

The State Detection page selects one object type and constructs a numerical feature space. The current implementation includes activity-count columns over object lifecycles, distinct related-object counts by object type, latest numeric object attributes, and one-hot categorical object attributes with fewer than 50 distinct latest

values. The feature table can be previewed and exported as CSV.

For execution-state abstraction, FLOWVAULT encodes sliding lifecycle windows, applies a deterministic two-component PCA implementation, and trains a deterministic SOM over the PCA coordinates. Each occupied SOM cell is treated as a discovered state. Consecutive windows of one object that move to another cell form transitions. Determinism supports repeatability for a fixed input and configuration, although stability under data perturbation still has to be evaluated.

SOM cells can be colored by assigned-window density or by a selected object attribute. Numeric attributes use cell averages, while categorical attributes use dominant-category counts. Selecting a cell opens a detailed view with a DFG for the selected object type and boundary-condition tabs for windows entering and exiting the cell. This implements the interpretability workflow described in Section 4.4.

The current feature table and automatic state pipeline instantiate Definitions 8 and 9 with aggregated windows. The interface can already apply SOM cells as the event-level `state` attribute and open the corresponding pattern and SA-OCDFG views. This materialization votes over every window containing an event, selecting the most frequent cell, breaking ties by latest window index and then cell order. Since such a window can end after the event, this is a retrospective labeling policy, distinct from the endpoint assignment in Section 4.4. Shared leading-object perspectives are also pooled at event level. Future work should expose this policy explicitly, add a prospective endpoint option, and export fitted models and complete provenance.

5.5 State-aware graphs, transition indicators, and patterns

FLOWVAULT exposes three graph computations. A single-type DFG follows the lifecycles of a selected object type. An OCDFG creates typed edges for all object types and adds typed START and END nodes. An SA-OCDFG uses the enriched state attribute, labels activities as Activity[State], and inserts CHANGE(*previous*, *next*) nodes when consecutive stateful lifecycle events change state. The graph data transfer object includes positioned nodes, routed edges, labels, weights, object-type colors, and node shapes, and the Angular interface renders it directly as SVG.

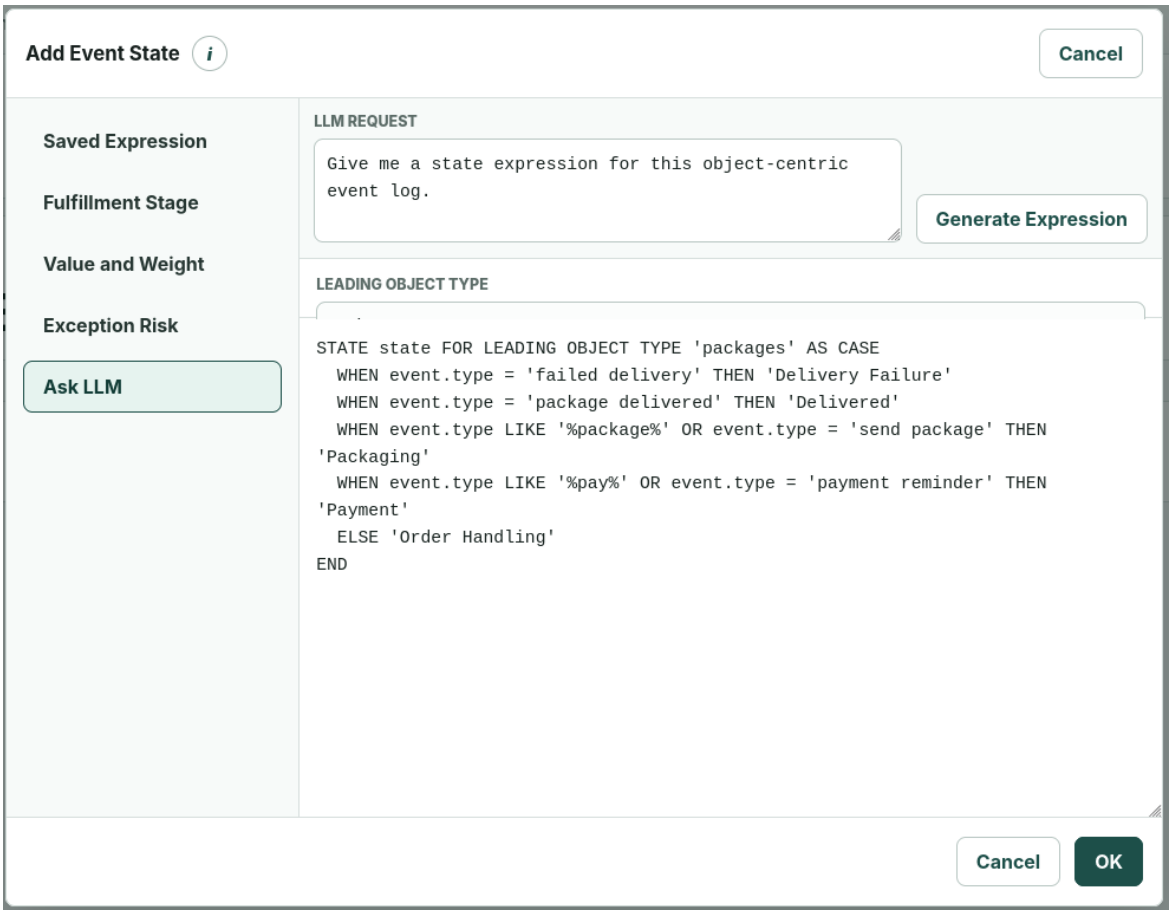

Fig. 14 LLM-assisted state definition in FLOWVAULT. The *Ask LLM* option accepts a natural-language request and generates a SQL-like state expression that the analyst can inspect and edit before applying it. The example proposes states for the `packages` leading object type.

The graph viewer supports object-type selection and minimum activity or path frequencies. Filtering is applied only when the analyst confirms the draft settings, which reduces accidental changes during exploration. State-specific node splitting can produce a large graph, so these controls are important for readability.

Transition KPI views report counts, object counts, and minimum, median, mean, and maximum durations for state changes. Dwell episodes are summarized by state. Recovery transitions and objects that remain in a final state for a long time can be inspected. The time perspective shows event-state frequencies over time buckets and a transition-performance spectrum for one-way or roundtrip transitions. These views operationalize the indicators in Section 4.8.1.

The Patterns page becomes available after event-level state enrichment, either from an expression or from applied SOM labels. For the selected leading type, consecutive lifecycle events with the same event state form event-state runs. These implement calendar episodes only in the event-driven special case; an update-only intervening episode is not represented by this implementation. Adjacent episodes form inter-state transition segments. Local graphs contain directly-follows, event-to-object-type, and object-object context edges. Structurally equal graphs are grouped and ranked by support and then control-flow mass. Analysts can switch between text and graph representations for intra-state and inter-state patterns.

The graph representation draws directly-follows and event-to-object-type edges with direction. Object-object context edges are displayed as undirected type links because they summarize co-participating object types rather than a causal or temporal order. This visual convention should not be interpreted as erasing the qualified direction of the original OCEL relation in the underlying data.

5.6 Lifecycle, association, and exploratory structural views

The lifecycle view presents the events of a selected object together with event attributes, state bands, stock points when available, and related objects. It is valuable for validating whether a state calendar and its transitions agree with the underlying evidence. Analysts can move from an aggregate transition or pattern to an individual lifecycle and inspect the exact sequence.

Correlation views connect object-level features with dominant states or state-related outcomes. They report feature associations and compare in-state with out-of-state means. Such views are useful for screening and interpretation, but they should not be treated as independent-effect estimates when features are correlated or objects are clustered.

The causal workbench supports a directed model with observable, latent, and outcome nodes and transformations such as identity, logarithm, and square root. The current fitting layer aggregates latent constructs and reports correlations and p-values on configured edges. This is best described as exploratory structural analysis. It does not by itself implement the identification, adjustment, estimation, and sensitivity procedures required for causal effect claims. The interface can nevertheless help analysts make assumptions explicit and prepare a model for a dedicated statistical environment.

5.7 Alignment between framework and implementation

Table 6 summarizes which formal concepts are implemented and where the current tool uses a more specific materialization. This distinction prevents the paper from presenting an abstract capability as already complete in software.

FLOWVAULT's main strengths are local browser execution, multi-format OCEL support, compact storage, deterministic core analyses, integrated state views, and inspectable transitions and patterns. Current limitations include event-level rather than incidence-level state materialization, limited confidence and provenance export, browser-memory constraints for very large logs, a fixed automatic-state pipeline, and exploratory rather than identified causal analysis. These limitations define concrete implementation work rather than weaknesses of the underlying state-aware abstraction.

The evaluation source revision, complete result snapshot, run configurations, and recorded build environment are identified in the Code and Data Availability statements. The observed input logs are also deposited separately on Zenodo [Berti, 2026b]. The repository result files use Git Large File Storage (Git LFS), which must be materialized when reproducing the analyses.

6 Evaluation

6.1 Evaluation objectives and common protocol

The controlled evaluation is organized around two complementary scenarios that are executed as reproducible, *oracle-backed* synthetic experiments: every state, transition, injected pattern, and outcome is generated together with an independent reference, so correctness can be measured instead of assumed. The experiments test state

Table 6: Alignment of the formal SA-OCPM framework with the current FLOWVAULT implementation.

Concept	Formal expectation	Current implementation	Constraint or next step
Base OCEL	Typed events, objects, updates, E2O and O2O relations	Compact Rust representation with lifecycle indexes and multi-format input or output	Browser memory limits should be benchmarked
State notion	Versioned context extractor and assignment with provenance	SQL-like CASE query (manual, preset, or LLM-assisted) or deterministic PCA and SOM pipeline	Export a complete provenance manifest
Incidence annotation	Separate state per event-leading-object incidence	One derived event attribute with existential object-condition semantics	Add incidence-level or explode materialization
State changes	Explicit records with source, target, phase, cause, and evidence	CHANGE nodes inferred between consecutive event-state labels	Represent object-update changes and phase directly
SA-OCDFG	State-aware labels, change nodes, typed lifecycle edges	Activity[State], CHANGE(*source*, *target*), typed START and END edges	Expose label and propagation policies
Episodes and patterns	Calendar episodes and segments with typed local graphs	Event-state runs, ranked by support then control-flow mass	Retain update-only and event-free episodes
Temporal indicators	Exposure, dwell, transitions, recovery, censoring	Transition KPIs, dwell summaries, spectra, and stuck objects	Add explicit censoring and observation-horizon controls
Automatic states	Temporally admissible feature windows and reproducible fitted model	Object features, windows, PCA, SOM, boundary views, and retrospective window-vote event labels	Add configuration export, confidence, and prospective endpoint materialization
Association and structural analysis	Valid estimands, uncertainty, and declared assumptions	Feature associations and exploratory causal workbench	Use dedicated statistical estimation for causal claims

and transition materialization, operational indicators, state-conditioned behavior, automatic-state diagnostics, prediction, pattern and conformance recovery, robustness, causal-pipeline bookkeeping, and computational performance. Section 6.5 complements these tests with evidence from previously reported real-life applications.

The common protocol addresses four evaluation questions:

- **EQ1.** Do expression-based states reproduce accepted operational definitions, and are automatic states stable and interpretable?
- **EQ2.** Does SA-OCPM reveal diagnostically useful behavior that is hidden or harder to obtain in state-agnostic baselines?
- **EQ3.** Do the findings remain under plausible changes to thresholds, window length, SOM size, filtering, missingness, and observation horizon?
- **EQ4.** How do runtime and memory change with events, objects, leading incidences, features, state count, and pattern diversity?

The experiments provide oracle-based evidence for **EQ1**, quantitative evidence for **EQ2** and **EQ3**, and native-core benchmark evidence for **EQ4**.

The experimental design combines independently materialized synthetic truth, retrospective log analysis, temporal and group holdouts, pre-specified perturbations, and controlled scale profiles. The generators use a fixed seed, and the source revision and run configurations are recorded. Object-bootstrap diagnostics resample leading objects. Temporal predictor holdouts purge training observations whose target horizon crosses the split and require test feature windows to start after it; group holdouts exclude complete object groups from predictor training. The same object may appear on both sides of a temporal split. These controls apply to predictor fitting, while the precomputed learned representations and oracle-derived state features impose the limitations stated in Section 7.5.

State validity is reported through several complementary measures. Expression-based states are checked for *branch coverage*, `Unknown` *exposure*, and agreement with independently computed reference labels, including boundary cases. Automatic states are assessed through occupancy, entropy, quantization quality, and agreement with simulator-defined execution regimes, using adjusted Rand index, normalized mutual information, and *purity*. Bootstrap and period diagnostics rescore the existing cell assignments; they do not test stability under retraining.

The quantitative comparisons use state-conditioned and state-agnostic next-activity distributions, independently generated state and transition references, injected pattern and conformance truth, and predictor feature families. The predictor baselines are static attributes, raw measurements, activity counts, state descriptors, object context, and their combination. All feature sets predict the same independently materialized state-defined targets. Complete-lifecycle variants, super variants, and state-agnostic segment miners are conceptual comparators in Section 2; no direct accuracy or analyst-performance comparison with these methods was run.

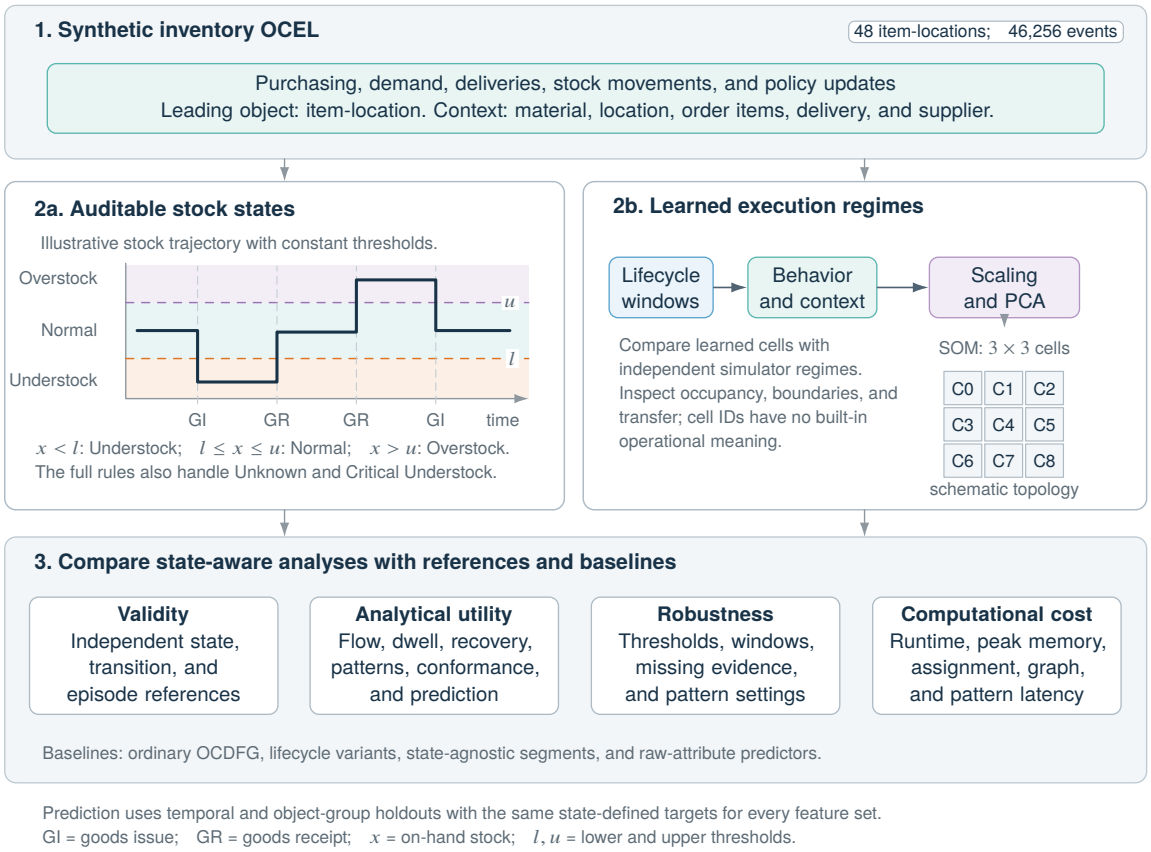

Fig. 15 Inventory-management evaluation scenario. An item-location OCEL feeds independently validated manual stock states and automatically discovered execution regimes. State-aware analytics are compared with state-agnostic baselines through oracle-based validity checks, robustness tests, prediction tasks, and performance benchmarks.

Technical benchmarks use controlled 75,000- and 300,000-event profiles and measure native import, state assignment, transition indicators, pattern extraction, and automatic-state computation. The paper-scale command additionally constructs the ordinary and state-aware graphs. Each profile has one measured repetition. Operating-system, memory, compiler, and source-revision information is recorded, but browser latency and independent effects of relation density, feature dimension, and pattern diversity are not measured.

6.2 Scenario A: inventory management

The first scenario studies inventory at the item-location level. The leading object type is Material-at-Location or an equivalent item-location object. Context objects include material, location, purchase order item, sales order item, delivery, supplier, and replenishment proposal. Events include stock-changing movements, purchase-order creation and change, goods receipt, goods issue, reservation, cancellation, transfer, and inventory adjustment.

Figure 15 summarizes the inventory-management evaluation design. The synthetic item-location OCEL represents purchasing, demand, deliveries, stock movements, and policy updates. An independently reconstructed stock trajectory supports the manual taxonomy of `Understock`, `Normal`, and `Overstock`. A parallel lifecycle-window pipeline discovers execution regimes automatically. Both state notions feed SA-OCPM analytics and are evaluated through *oracle-based validity checks*, robustness perturbations, prediction tasks, and computational benchmarks.

The manual state notion uses time-varying on-hand stock and policy thresholds. Let $x_o(t)$ be on-hand stock, $l_o(t)$ the lower or safety threshold, and $u_o(t)$ the upper or maximum threshold. A basic rule assigns `Understock` when $x_o(t) < l_o(t)$, `Normal` when $l_o(t) \leq x_o(t) \leq u_o(t)$, and `Overstock` when $x_o(t) > u_o(t)$. `Unknown` is assigned when the trajectory or thresholds cannot be established. A more decision-oriented extension distinguishes `Critical Understock` when confirmed demand exceeds usable stock plus timely inbound replenishment.

Listing 2: Expression-based state definition for Scenario A

```
STATE state FOR LEADING OBJECT TYPE 'ItemLocation' AS CASE
  WHEN event.data_complete = false THEN 'Unknown'
  WHEN event.critical_understock = true THEN 'Critical Understock'
  WHEN event.on_hand_after < event.lower_threshold THEN 'Understock'
  WHEN event.on_hand_after > event.upper_threshold THEN 'Overstock'
  ELSE 'Normal'
END
```

The reference labels are computed by a separate Python function over the simulator's post-event inventory state, without using FLOWVAULT's expression parser or evaluator. This checks the query implementation against a separately implemented assignment, but both use the same simulator semantics. It does not independently validate stock reconstruction or the operational taxonomy. On organizational data, disagreements would also need to be traced to missing observations, trajectory reconstruction, policy versions, and query errors.

The implemented automatic state notion aggregates activity counts and distinct related-object counts over lifecycle windows and includes leading-object attributes valid at each window endpoint. Static object-to-object links in the completed log also contribute context. It does not add explicit elapsed-time variables or event-level stock summaries to the SOM input. Cells can be inspected through density, attribute overlays, cell DFGs, and entry and exit windows. The experiment scores them against simulator execution regimes such as demand surge, replenishment in transit, or receipt-driven excess; those reference names are evaluation labels rather than expert-validated cell descriptions.

The automatic and manual notions answer related but non-identical questions. The manual notion provides a policy state based on stock level, whereas the automatic notion may discover execution regimes that precede or cut across stock states. The evaluation therefore reports both *alignment* and *incremental utility*. For example, a learned state is valuable if it predicts entry into `Understock` earlier than the stock threshold while remaining interpretable and stable.

The main analytical questions are as follows. Which intra-state patterns dominate prolonged `Understock` and `Overstock`? Which inter-state patterns precede `Normal` to `Understock` and `Normal` to `Overstock`? How do dwell and recovery times vary by supplier, location, product class, and replenishment policy? Which related objects and activities distinguish rapid recovery from persistent episodes? Does the SA-OCDFG reveal state-specific waiting times or object interactions that the ordinary OCDFG merges?

Table 7 specifies the data, comparisons, measures, and expected evidential contribution for the inventory scenario.

The inventory robustness runs vary stock thresholds, policy-update timing, event and relationship deletion, event-attribute missingness, timestamp jitter, stock-adjustment omission, window length, SOM size, and pattern radius. They use a small deterministic validation fixture, described in Section 7.6. Observation-horizon changes, minimum-duration smoothing, and feature-removal ablations were not executed.

6.3 Scenario B: manufacturing and predictive maintenance

The second scenario evaluates a different operational setting in which state is not reducible to one stock measure. The leading object is a machine or production asset. Context objects include production order, operation, work order, component, material lot, alarm, inspection, operator, and maintenance team. Events include operation start

Table 7: Inventory-management evaluation protocol.

Item	Design	Measures	Comparison
Leading perspective	Item-location lifecycle with purchase, sales, delivery, supplier, and location context	Coverage, lifecycle length, relation density	Same OCEL for all conditions
Manual state	`Understock`, `Normal`, `Overstock`, `Critical Understock`, and `Unknown`	Agreement with independent trajectory, boundary review, `Unknown` exposure	Validated reference computation
Automatic state	Window activity counts, related-object counts, and time-valid object attributes; PCA and SOM	Occupancy, stability, entropy, interpretability, alignment, early warning	Alternative window and grid settings
Diagnostic analysis	SA-OCDFG, dwell, recovery, intra-state and inter-state patterns	Pattern support, transition localization, dwell and recovery discrimination	State-agnostic next-activity distributions and injected pattern truth
Robustness	Vary thresholds, timing, missingness, window and grid settings, and pattern radius on a small fixture	Finding stability and transition-time drift	Predefined perturbation grid
Prediction	Early warning of undesirable stock conditions	Precision-recall, lead time, false alerts, missed episodes	Feature ablations without state
Performance	Controlled 75k/300k event profiles	Import, peak memory, assignment, transition, pattern, and discovery time	Fixed hardware and software commit

and completion, mode change, alarm, quality inspection, defect detection, maintenance request, maintenance start, part replacement, test, restart, and maintenance completion.

The manual state notion combines machine mode, alarm-derived flags, and maintenance status. Its taxonomy is `Running`, `Idle`, `Setup`, `Degraded`, `Down`, `Recovery`, `Quality Hold`, and `Unknown`. The generator maintains the observed degradation flag using hysteresis and a stable-running requirement for recovery. The query reads these derived flags from event attributes. Sensor snapshots and alarm events provide the observations used in the experiment; changes occurring solely between recorded events are supported by the abstract framework but are not tested here.

Listing 3: Expression-based state definition for Scenario B

```
STATE state FOR LEADING OBJECT TYPE 'Machine' AS CASE
  WHEN event.data_complete = false THEN 'Unknown'
  WHEN event.down_active = true OR event.mode = 'DOWN' THEN 'Down'
  WHEN event.quality_hold_active = true THEN 'Quality Hold'
  WHEN event.recovery_active = true THEN 'Recovery'
  WHEN event.mode = 'SETUP' THEN 'Setup'
  WHEN event.degraded_latched = true THEN 'Degraded'
  WHEN event.mode = 'RUNNING' THEN 'Running'
  ELSE 'Idle'
END
```

The automatic notion uses the same window encoder as inventory: activity counts, related-object counts, and time-valid leading-object attributes. Alarm activity types and related work and production objects can enter through these features, but explicit elapsed-time features and event-level sensor summaries are not added to this SOM representation. PCA and SOM identify recurring execution regimes. Cell explanations use activity distributions, attribute overlays, object context, and boundary windows. Learned cells are compared with simulator execution regimes, which are separate from the operational state taxonomy used for transition and recovery analysis.

Figure 16 summarizes the manufacturing and predictive-maintenance evaluation. A machine lifecycle is linked to production orders, alarms, inspections, work orders, components, and operators. Aligned manual and learned calendars expose the windows around the `Degraded` to `Down` and `Down` to `Recovery` transitions. State-aware patterns, performance indicators, and predictions are then assessed through robustness checks and computational benchmarks.

The principal questions are which patterns precede `Degraded` to `Down`, which coordination patterns distinguish quick from slow recovery, which activities recur while a machine remains `Down`, and whether `Quality Hold` episodes are preceded by specific machine and production-order interactions. A state-aware analysis can also test whether maintenance starts within a required time after a critical alarm and whether restart occurs only after inspection or test evidence.

Table 8 summarizes the design. The scenario deliberately includes both process events and measurements so that the evaluation can test whether SA-OCPM connects operational behavior with machine condition without reducing the problem to pure time-series classification.

Table 8: Manufacturing and predictive-maintenance evaluation protocol.

Item	Design	Measures	Comparison
Leading perspective	Machine lifecycle with production orders, work orders, alarms, components, inspections, and operators	Coverage, event and update density, machine heterogeneity	Same OCEL for all conditions
Manual state	`Running`, `Idle`, `Setup`, `Degraded`, `Down`, `Recovery`, `Quality Hold`, `Unknown`	Agreement with mode and maintenance records, boundary review, chattering	Independent operational labels
Automatic state	Window activity counts, related-object counts, and time-valid machine attributes	Stability, transfer, occupancy, interpretability, alignment	Window and grid perturbations on a small fixture
Diagnostic analysis	Patterns into `Down` and `Recovery`, dwell, recurrence, response conformance	Pattern support, transition localization, recovery discrimination	State-agnostic next-activity distributions and injected violations and patterns
Prediction	`Down` within horizon and time to stable recovery	Precision-recall, calibration, lead time, false alerts, time error	Raw features without state and state without object context
Robustness	Vary alarm timing, sensor missingness, window length, and grid size	Finding stability and label drift	Predefined perturbation grid
Performance	Controlled 75k/300k event profiles with low-dimensional features	Runtime, memory	Fixed hardware and software commit

Retrospective validation compares state assignments with independently maintained mode and maintenance records, including transitions that are false positives or false negatives under the manual rule. Automatic-state agreement is examined across production periods and object-bootstrap samples while keeping the fitted map fixed. *Label transfer* fits the interpretation of each cell on some machine groups and evaluates it on others. Retraining and transferring the complete discovery pipeline remain future validation steps.

The prediction experiment estimates entry into `Down` within a future horizon and time to recovery. Models compare state-derived, raw, static, activity-count, object-context, and combined representations. Temporal and machine-group splits constrain predictor fitting; they do not remove leakage from the previously fitted SOM and simulator-based cell interpretation. Metrics include precision-recall, calibration, warning lead time, false alerts, and recovery-time error. The feature-family comparisons do not isolate the incremental effect of object context because no full-minus-context model was fitted.

6.4 Cross-scenario hypotheses and assessment criteria

The two scenarios test complementary aspects. Inventory management offers a strong rule-based reference state and interpretable economic thresholds, whereas manufacturing offers *multi-signal* states, explicit sensor and alarm observations, and safety-relevant transitions. A method that works only when one clean numeric attribute defines the state would perform well in the first scenario but would not establish *generality*.

Table 9 lists the pre-specified cross-scenario hypotheses together with the evidence required to support or limit them.

Quantitative results are interpreted with uncertainty in mind. Bootstrap resampling occurs at the leading-object or cluster level, and multiple comparisons across states or patterns are either controlled or explicitly labeled *exploratory*.

The completed predictor comparisons use five feature groups—static attributes, raw measurements, activity counts, state descriptors (including SOM descriptors), and object context—plus their concatenation. They do not include one-factor removal of state-change nodes, state-conditioned labels, object context from an otherwise complete model, or boundary windows, nor a separately validated hybrid taxonomy. These remain useful follow-up ablations. Pattern retrieval is evaluated on the behavior-log projection, while state validity and

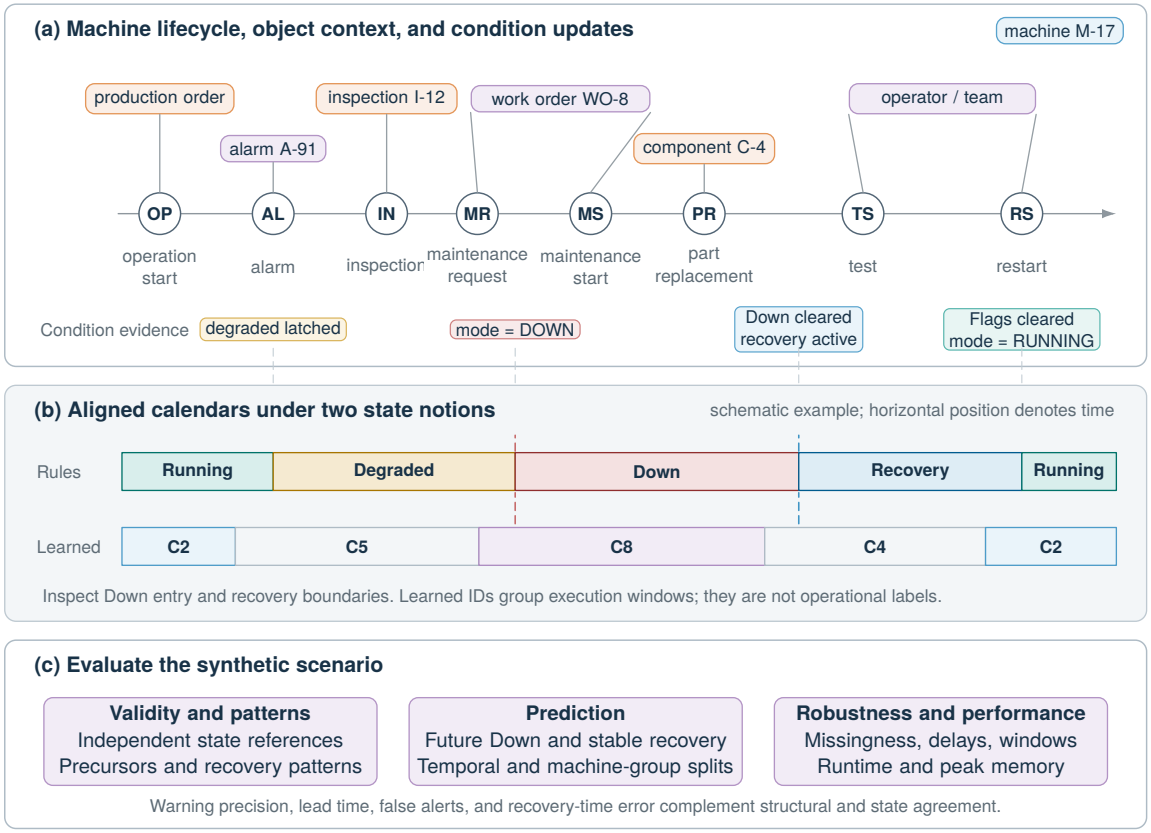

Fig. 16 Manufacturing and predictive-maintenance evaluation scenario. A schematic machine lifecycle illustrates rule-based and learned state calendars, transition-pattern and recovery analyses, next-state and recovery prediction, robustness checks, and performance benchmarks.

Table 9: Cross-scenario hypotheses and required evidence.

ID	Hypothesis	Primary evidence	Falsifying or limiting evidence
H1	Manual and automatic state notions provide complementary information.	Stable learned states with incremental prediction or diagnosis beyond manual labels	Learned cells are unstable, opaque, or duplicate the manual taxonomy without benefit
H2	State-determined patterns localize mechanisms of persistence and transition.	Distinct intra-state and inter-state patterns recur with substantial support and match injected mechanisms	Patterns are trivial, sparse, or no more useful than complete variants
H3	Object context contributes beyond state and activity order alone.	Removing context features reduces predictive or diagnostic quality in ablations	Context features add complexity without measurable benefit
H4	The implementation supports realistic interactive analysis.	Acceptable import, state, graph, and pattern latency on target logs	Peak memory or latency prevents completion on representative data
H5	Key findings are robust to plausible state and preprocessing choices.	Same substantive conclusions across pre-specified sensitivity ranges	Conclusions reverse under small threshold, window, or missingness changes

automatic-state diagnostics use the full observed logs.

Reproducibility is supported by the synthetic OCELs, extraction documentation, state definitions, model configurations, the software commit, and the analysis scripts. The evaluation report distinguishes *pre-specified* analyses from *exploratory* follow-up.

The most important success criterion is not that every learned state matches a manual label. Success means that the state notion is *reproducible*, understandable enough for its intended use, *temporally valid*, and demonstrably improves a relevant analytical task without creating unacceptable alert burden or computational cost.

6.5 Evidence from real-life applications

SA-OCPM has also been applied successfully to real-life operational data in the studies on which this framework builds. In the original study, Kretzschmann et al. [2026b] implemented state-aware enrichment in Celonis and analyzed a European pet retailer's warehouse data from December 2022 to April 2024, covering 9,648 materials. Explicit stock states and transitions exposed shortages that persisted despite goods receipts, demand without corresponding purchase orders, and replenishment during overstock. State-aware paths and recovery times connected these conditions to the surrounding purchasing behavior. This constitutes evidence of practical diagnostic applicability on an organizational log.

Subsequent studies demonstrate additional parts of the analysis pipeline on real retail data. Berti et al. [2026c] derived execution-state trajectories from inventory and planning records using lifecycle windows, PCA, and a SOM. Including stock information supported the interpretation of learned regimes and a policy matrix linking states and boundary conditions to candidate actions. Kretzschmann et al. [2026a] reported automated intra-state and inter-state pattern detection on a real-life pet-retail OCEL using companion evaluation scripts, obtaining recurring object-interaction patterns associated with understock and overstock. Berti et al. [2026a] reconstructed item-location stock trajectories and used time in understock and overstock as outcomes in a structural equation model; the real-data analysis related undesirable exposure to ordering irregularity and reorder-trigger execution. These applications demonstrate that the state perspective supports concrete diagnostics and interpretable outcomes beyond synthetic logs.

The evidence is concentrated in retail inventory and should be read as complementary to the controlled experiments reported here. The earlier applications establish feasibility and report useful diagnostic findings; they do not establish cross-domain generalization, causal intervention effects, or the reliability of every component of the unified framework. Conversely, the synthetic scenarios provide independently materialized references for states, transitions, and outcomes, which are generally unavailable in organizational data. No new organizational dataset is analyzed in the present assessment. The real-life reports and the synthetic tests therefore address different parts of the evidence needed for adoption.

7 Assessment

7.1 Execution scope, data quality, and oracle agreement

The two scenarios were executed as reproducible, oracle-backed synthetic experiments rather than as demonstrations with hand-selected toy traces. Both generators used seed 20260826, and generation, validation, and analysis were executed against FLOWVAULT commit `b16000afd6a5271a6f822c5bd8118a1a01fcc74e`. The inventory run covers two years and contains 46,256 events, 18,506 objects, and 187,191 event-to-object relations. The manufacturing run covers approximately fifteen months and contains 43,458 events, 3,578 objects, and 92,581 event-to-object relations. Every event was related to exactly one leading object in the selected

perspective, while the additional relations retained the surrounding object context. Table 10 summarizes the scale after state enrichment.

Schema validation, relationship-reference validation, semantic checks, and the scenario-specific conservation or branch-coverage assertions passed for both logs. Relative to the independently materialized synthetic oracle, the expression-based state queries achieved complete coverage, accuracy and macro-F1 of 1.0,[1] temporal episode intersection-over-union of 1.0,[2] and transition precision and recall of 1.0. All 10,833 inventory and 11,196 manufacturing transitions were matched with zero median timestamp error.[3] These perfect values should be interpreted as an internal correctness result: they show agreement for the tested event-driven, single-leading-object materialization. They do not test disagreeing states on shared leading events or changes occurring only between event observations. They do not establish that the same expressions would be correct for an organizational log whose measurements, policies, and recording practices are uncertain.

The oracle comparison also exposes two issues that are hidden by accuracy alone. First, `Unknown` is substantial in both experiments because incomplete observations were deliberately injected. Second, 23.76% of inventory episodes and 12.67% of manufacturing episodes lasted less than one minute. Some short episodes are legitimate event-time changes, but others are caused by brief `Unknown` intervals or rapid switching around boundaries. Consequently, production use should report `Unknown` separately and should consider hysteresis, minimum-duration rules, or confidence-qualified transitions rather than silently smoothing the calendar.

7.2 Operational findings in the inventory scenario

The state calendar changes the interpretation of the inventory log in two ways. Event frequency identifies `Critical Understock` as the dominant event label, whereas object-time shows that `Normal` accounts for slightly more total exposure than `Critical Understock`. This difference is visible in Table 11: state-aware duration measures therefore avoid equating a high event rate with a long-lasting condition. `Critical Understock`, `Understock`, and `Overstock` episodes all have a median duration of approximately two days, while the median `Normal` episode lasts approximately three days. The median time from entry into a contiguous shortage run to the next `Normal` episode is 6,490.7 minutes, or 4.51 days. Here shortage combines `Understock` and `Critical Understock`; an intervening other state resets this calculation. The evaluator imposes no minimum duration in `Normal`, so this is a first-return measure rather than stable recovery.

The recurrence analysis identifies 114 recovery opportunities. A new shortage occurs within 7 days in 9.65% of them, within 14 days in 24.56%, and within 30 days in 39.47%. At the end of the observation horizon, 23 of the 48 item-location objects end in a right-censored shortage episode. This reported count imposes no minimum-duration threshold. The recurrence fractions use all observed direct shortage-to-`Normal` returns and do not adjust for incomplete follow-up near the horizon. They are descriptive observed fractions, not censoring-adjusted recurrence risks. These results illustrate the distinction among first return, subsequent recurrence, and the final operational queue: an object can reach `Normal` quickly and still return to shortage soon afterwards. The exposure ranking, 4.51-day median first-return time, recurrence rates, and stuck-object count all depend on the constructed state calendar and its declared policies. They cannot be obtained from the ordinary OCDFG or activity-based variants alone: event counts and path durations do not specify when shortage begins, which return to `Normal` is counted, or whether an object remains in shortage at the horizon.

Figure 17 shows the corresponding transition matrix and duration summaries in FLOWVAULT. The most frequent changes are `Unknown` to `Critical Understock` (3,379) and `Critical Understock` to `Unknown` (3,358). Because `Unknown` is assigned when `event.data_complete=false`, these transitions primarily reveal observation gaps rather than physical stock movements. Direct recoveries remain visible separately: `Critical Understock` to `Normal` occurs 69 times with a median of 4.67 days, `Understock` to `Normal` occurs 45 times with a median of 3.19 days, and `Overstock` to `Normal` occurs 34 times with a median of 24 hours. Treating

[1] Macro-F1 is the unweighted mean of the per-state F1 scores, where each F1 score is the harmonic mean of precision and recall. It therefore gives every state equal weight regardless of its frequency.

[2] Temporal episode intersection-over-union is the duration for which the derived and reference calendars assign the same object to the same state, divided by the union of their object-state durations.

[3] Transition precision divides tolerance-matched transitions by all derived transitions, transition recall divides them by all reference transitions, and timestamp error is the absolute time difference within each matched pair; the reported value is its median.

Table 10: Scale and state-enrichment outcomes for the two paper experiments. `Unknown` here is the fraction of events for which the required state evidence was intentionally incomplete.

Scenario	Events	Objects	Leading	E2O	O2O	States	Transitions	Unknown
Inventory	46,256	18,506	48	187,191	18,551	5	10,833	12.94%
Manufacturing	43,458	3,578	8	92,581	3,554	8	11,196	10.78%

Table 11: Selected state-episode results. Exposure aggregates object-time over all leading objects and is therefore larger than the wall-clock observation horizon.

Scenario	State	Episodes	Median dwell	Total exposure
Inventory	`Critical Understock`	3,489	2.00 d	12,816.14 d
Inventory	`Normal`	1,431	3.00 d	13,364.73 d
Inventory	`Understock`	572	2.00 d	2,754.18 d
Inventory	`Overstock`	185	2.00 d	978.75 d
Manufacturing	`Running`	5,096	8.01 h	2,543.28 d
Manufacturing	`Degraded`	478	1.02 min	94.04 d
Manufacturing	`Down`	282	30.00 h	508.37 d
Manufacturing	`Recovery`	310	4.00 h	62.40 d

`Unknown` as a first-class state prevents those gaps from being misread as instantaneous recovery or deterioration.

State conditioning also changes the control-flow representation. The inventory leading-object lifecycles contain 158 distinct (current activity, next activity, current post-state) combinations. This statistic excludes START, END, and CHANGE markers and is not the edge count of the full SA-OCDFG. The conditional mutual information between state and the next activity, given the current activity, is 0.118, and the weighted Jensen–Shannon divergence among state-conditioned next-activity distributions is 0.035.[4] The values are not large enough to claim that state dominates control flow, but they are consistently nonzero: the same activity edge is used with measurably different continuation distributions across stock regimes. Figure 18 illustrates one recurring inter-state pattern. The explicit state-change node separates behavior before and after the `Critical Understock` to `Unknown` boundary, while the lower layer preserves the ItemLocation, Location, Material, and sales-order context.

7.3 Operational findings in the manufacturing scenario

The manufacturing experiment shows that the same abstractions remain meaningful when state is determined by several operational signals rather than one stock trajectory. `Running` dominates exposure, but the 282 `Down` episodes still accumulate 508.37 machine-days and have a median duration of approximately 30 hours. `Recovery` episodes have a median duration of four hours. The transition sequence is highly structured: `Degraded` to `Down` occurs 273 times, `Down` to `Recovery` 277 times, and `Recovery` to `Running` 250 times, whereas a direct `Running` to `Down` transition occurs only four times. An ordinary OCDFG can merge the corresponding behavior into high-frequency activity edges. SA-OCPM makes the operational pathway explicit: the 508.37 machine-days in `Down` and the distinction between downtime and recovery require the machine-specific state calendar, not merely the durations between maintenance activities.

Figure 19 shows the transition and dwell-time views. The frequent `Running` to `Unknown` and `Unknown` to `Running` transitions reflect incomplete sensor or mode evidence, while the chain from `Degraded` through `Down` and `Recovery` back to `Running` remains visible as a separate operational pathway. Two machines end the horizon in a right-censored `Down` episode, without a minimum-duration threshold in the reported count. The manufacturing leading-object lifecycles contain 105 distinct state-conditioned activity-pair combinations under the same definition, with conditional mutual information 0.089 and weighted Jensen–Shannon divergence 0.020. As in inventory, state contributes additional but not exclusive information about the next activity.

7.4 Automatic state discovery

This experiment asks whether groups of similar execution windows provide useful state descriptions without handwritten state expressions. The pre-specified 3×3 SOM uses three-event windows for inventory and four-event windows for manufacturing. PCA fitting and ten SOM training epochs use a deterministic, evenly spaced sample of 2,000 windows per scenario; all complete windows are then assigned to cells. The 51 and 60 input columns are aggregated window features, not per-event vectors multiplied by the window length. The results support inspecting learned cells for recurring regimes, but do not establish a reusable operational state taxonomy.

Agreement with known execution regimes. The reference labels come from the simulator's recorded execution regimes, such as `Supplier Delay` and `Bearing Degradation`. These describe the mechanisms generating behavior and are distinct from the rule-defined operational states, such as `Understock` and `Down`.

[4] Conditional mutual information measures, in bits, how much knowing the state reduces uncertainty about the next activity after the current activity is known. The weighted Jensen–Shannon divergence compares each state-specific next-activity distribution with the corresponding state-agnostic distribution and averages the bounded, symmetric divergences by observed activity and state frequency.

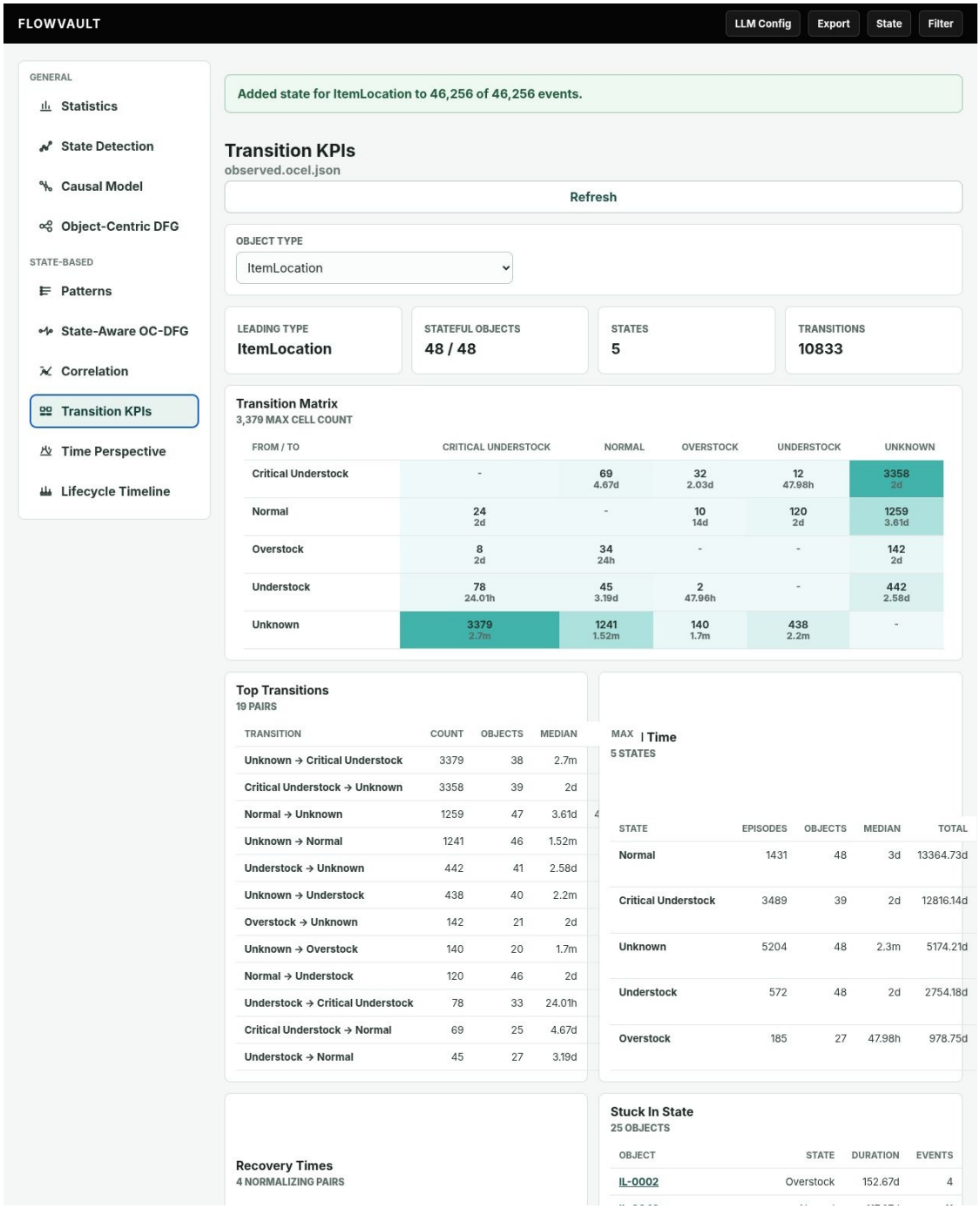

Fig. 17 Inventory transition-KPI view after enriching all 46,256 events. The cropped production screenshot retains the state-transition matrix, frequent transition pairs, episode durations, and the leading part of the recovery and stuck-state summaries.

Each window receives the regime that occurs most often among its events. The SOM is not trained to reproduce these labels; agreement is an external diagnostic of what its cells capture.

Table 12 separates this agreement from the warning results. *Purity* is the fraction of all windows that agree with the most common reference regime in their own cell. Its values, 82.7% and 84.7%, indicate a dominant regime within much of the partition. Because this is a window-weighted measure, large cells contribute more than small ones; it does not mean that every cell is equally coherent.

Normalized mutual information (NMI) measures agreement between the complete cell partition and the reference regimes: 0 indicates no shared information and 1 indicates identical partitions, apart from label names. The scores of 0.225 and 0.075 show that high purity does not translate into a close reconstruction of the regime structure. The *adjusted Rand index* (ARI), which measures pairwise grouping agreement after correcting for

Table 12: Automatic-state diagnostics for the 3 × 3 SOM. Purity, NMI, ARI, and resampling use the majority simulator regime within each window; label transfer uses the endpoint regime. Resampling and transfer use 2,000 selected windows per scenario with the SOM fixed. Warning results concern future adverse episodes, with horizons of seven days for inventory and 24 hours for manufacturing.

Measure	Inventory	Manufacturing
Data and agreement with simulator regimes		
Lifecycle windows	46,160	43,434
Input features	51	60
Within-cell majority share (purity)	82.7%	84.7%
Overall regime agreement (NMI)	0.225	0.075
Chance-adjusted grouping agreement (ARI)	0.106	0.079
Resampling and transfer of cell labels		
Regime agreement after resampling (mean NMI)	0.267	0.085
Held-out group label recall (balanced accuracy)	25.0–64.8%	13.2–14.3%
Warnings from entry into risk cells		
Correct warnings (precision)	49.7%	9.7%
Adverse episodes warned (recall)	49.4%	48.6%
Advance warning for matched episodes (median)	4.0 days	14.0 hours
False warnings per leading object per week	0.406	2.48

chance, confirms this result at 0.106 and 0.079; 0 is the chance reference and 1 is perfect agreement. A dominant regime can account for many windows even when less frequent regimes are mixed together or split across cells.

What resampling and transfer establish. The bootstrap diagnostic resamples leading objects and recomputes NMI between their existing cell assignments and reference regimes. Across three repetitions on a 2,000-window diagnostic sample, mean NMI remains low at 0.267 and 0.085. This checks regime agreement under changes in the sampled object mix. It does not retrain the SOM, so these values do not measure whether a newly fitted map would reproduce the original cells. Stability after refitting on resampled objects remains untested.

The transfer diagnostic also keeps the map fixed. It learns a dominant regime label for each cell from the non-held-out groups, then checks those labels on the held-out group, using the reference regime at each window's endpoint. *Balanced accuracy* averages recall across the reference regimes, giving rare regimes the same weight

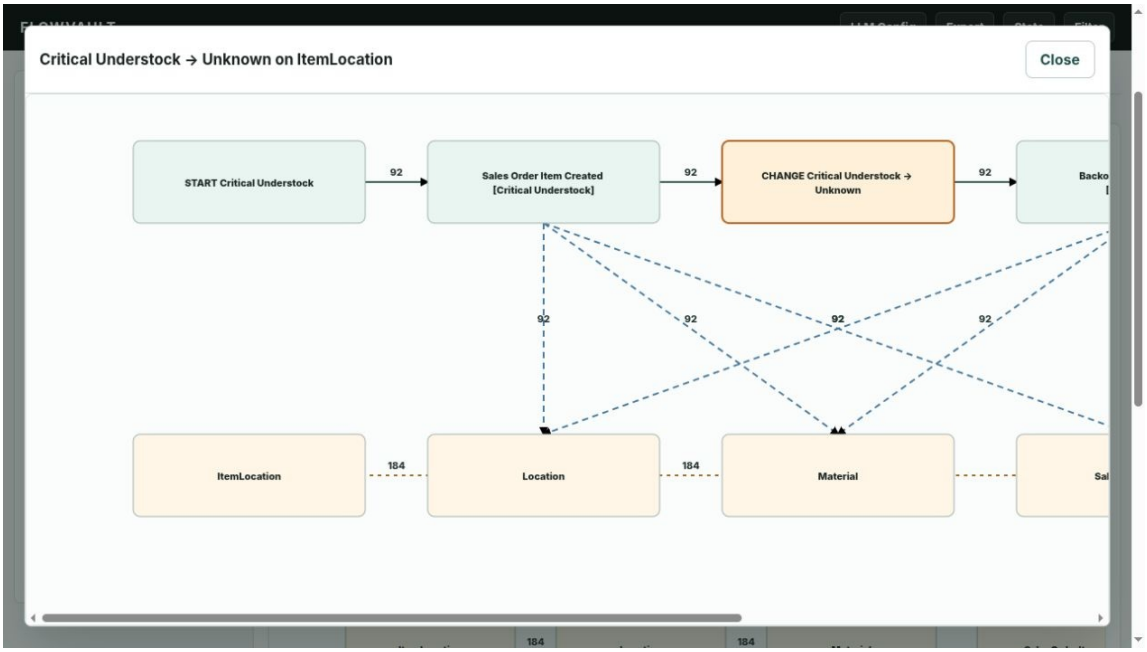

Fig. 18 Representative inventory inter-state pattern in FLOWVAULT. The pattern has support 92 and places the explicit `Critical Understock` to `Unknown` change between state-conditioned activities while retaining participating object types. The screenshot shows the visible portion of the horizontally scrollable graph.

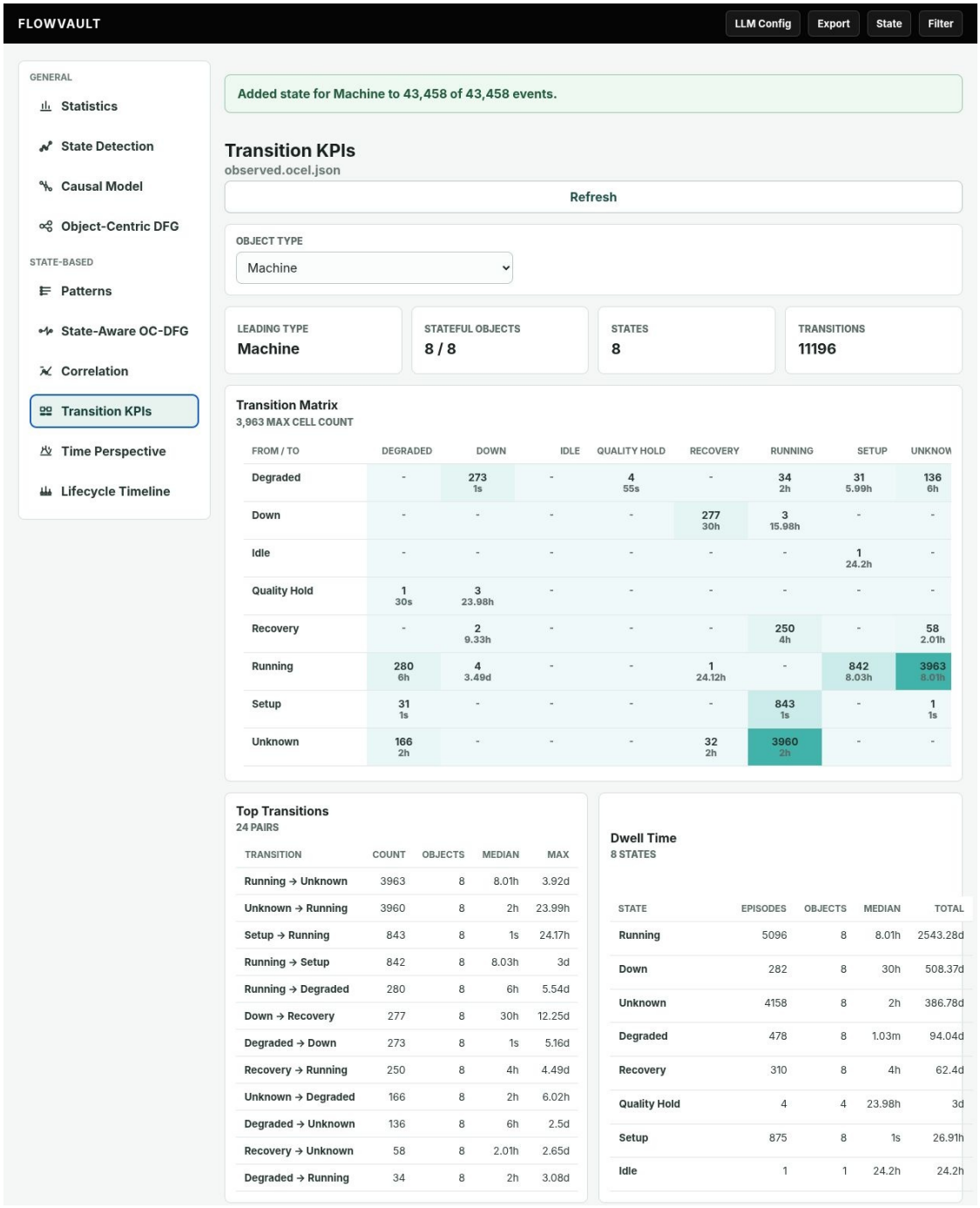

Fig. 19 Manufacturing transition-KPI view after enriching all 43,458 events. The transition matrix exposes the frequent `Degraded` to `Down`, `Down` to `Recovery`, and `Recovery` to `Running` pathway alongside `Unknown`-related observation gaps; the lower panels summarize dwell time and current long-running states.

as common ones. Scores range from 25.0% to 64.8% across inventory material and location classes, and from 13.2% to 14.3% across manufacturing machine families and sites. Thus even the interpretation attached to an existing cell transfers unevenly. A stronger test would fit the complete discovery pipeline on training objects and evaluate it on unseen objects.

Early-warning quality. Cells are associated with simulator regimes, and entry into a selected risk cell triggers a warning. The target is an `Understock` or `Critical Understock` episode within seven days for inventory, or a `Down` episode within 24 hours for manufacturing. *Precision* is the share of warnings followed by a matching adverse episode; *recall* is the share of adverse episodes preceded by a warning. Median lead time measures how far in advance matched warnings arrive, while false warnings per object-week express the review burden.

Inventory warnings identify about half of the adverse episodes, and about half of their alerts are correct. The median advance warning is four days, with 0.406 false warnings per item-location per week. This suggests a possible use in prioritizing inspections, subject to the cost of reviewing false alarms. Manufacturing reaches similar recall, but only about one warning in ten is correct, with 2.48 false warnings per machine-week despite a median lead time of 14 hours. Useful lead time alone therefore does not make a warning useful in practice. These are retrospective diagnostics using simulator-assisted cell interpretation, rather than a prospective deployment test.

Figure 20 documents a separate browser run on the manufacturing log. Its largest cell contains 30,777 windows, whereas the budgeted command-line run underlying Table 12 has 19,143 windows in its largest cell. The screenshot therefore illustrates the interface and a different fitted map; it is not the occupancy record for the tabulated diagnostics.

7.5 Prediction, patterns, conformance, and causal sanity checks

The computations reported in this subsection take place *outside the FLOWVAULT application*, in an external evaluation workflow based on a state-enriched OCEL exported by FLOWVAULT. The export carries the events, attributes, object relationships, and derived state annotations needed for further analysis. Separate scripts fit prediction models, score pattern retrieval, evaluate conformance rules, and run the causal sanity checks. Current-state, dwell, and transition-count predictor features are read from the simulator reference sidecars; their event labels agree exactly with the exported expression-based states in these runs. Pattern extraction reuses the FLOWVAULT core through its command-line interface; the comparative evaluation is performed externally. Simulator reference files supply the known future outcomes, planted patterns, and violations used for scoring. The causal checks additionally require simulator-supplied intervention pairs.

These experiments address different questions. Prediction tests whether state-derived inputs help anticipate future conditions. Pattern and conformance checks test whether relevant behavior and policy violations can be recovered. The causal check tests effect calculation when both intervention outcomes are known. Success in one task does not establish success in the others.

Prediction and score interpretation. The *temporal holdout* trains the prediction models on earlier observations and tests them on later ones. Each task uses at most 1,000 decision samples, selected deterministically with proportional label stratification and temporal spread before splitting. Predictor activity and object-context features use the latest eight lifecycle events. All feature sets predict the same state-defined outcomes. Table 13 compares three representations: *state features* include the current reference state, dwell time, transition counts, and SOM cell and regime descriptors; *raw attributes* contain the observed measurements without state-derived inputs; and *all features* combine these with static attributes, activity counts, and related-object context. Each

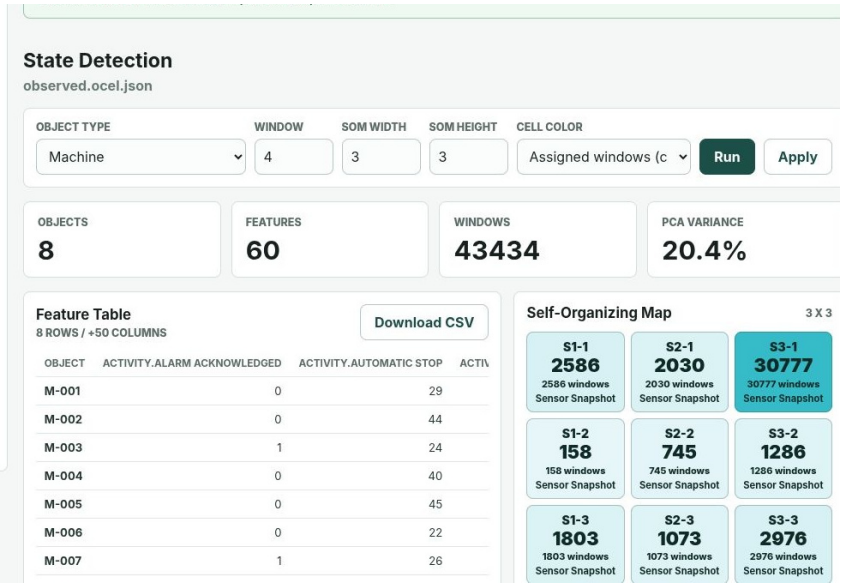

Fig. 20 Manufacturing automatic-state workspace (cropped to the analysis panel). The view shows the 60-feature lifecycle representation, 43,434 windows, and the 3×3 SOM occupancy. This separate browser fit differs from the budgeted command-line fit used for the numerical diagnostics. Cell identifiers are execution-cluster labels, not operational states such as `Running` or `Down`.

score is the higher observed AUPRC of logistic regression and histogram gradient boosting for that feature set. Selecting the higher test score makes this an exploratory comparison rather than an estimate for a model selected independently of the test set.

AUPRC, the area under the precision-recall curve, is computed here as average precision: precision is averaged over score thresholds, weighted by the increase in recall. Higher values mean that future positive cases tend to appear earlier in the ranked predictions; 1 denotes perfect ranking. The fraction of positive cases in the sampled test set provides the no-skill reference for that comparison. For example, 36.3% of inventory test cases enter `Understock` within seven days, giving a reference score of 0.363. AUPRC is not the fraction of correct alerts at a chosen threshold, and scores from tasks with different positive rates should not be compared as if they had the same difficulty.

The inventory results show predictive signal in the state features, with higher scores from raw attributes on both tasks. Combining all inputs gives the highest scores, although the gains over raw attributes are small, from 0.927 to 0.938 and from 0.558 to 0.581. In manufacturing, combining inputs helps at the 24-hour horizon, while state features alone give the highest score at four hours. The latter comparison is based on only four positive test cases, so its apparent advantage requires confirmation. The separate static-only and activity-count baselines score below the state-feature representation on all four tasks, as reported in the companion results.

These comparisons evaluate the supplied representations. The SOM assignments and their simulator-based regime mapping were constructed before the prediction splits, rather than fitted anew within each training set. The window encoder also uses object-to-object links from the completed log without timestamps establishing when they became available. Current-state histories come from the reference sidecars; exact expression agreement makes them equivalent to the derived event-state histories in these two runs, but does not test prediction under state-estimation error. The results therefore do not establish prospective performance of the complete state-discovery and prediction pipeline. The combined representation also adds several feature families at once, so its improvement over raw attributes cannot be attributed solely to states or object context.

Prediction on other object groups. A *group holdout* tests the fitted predictors on object groups excluded from predictor training. The ranking of feature sets changes: raw attributes outperform all features for inventory `Critical Understock` (0.614 versus 0.559), manufacturing `Down` within 24 hours (0.330 versus 0.274), and recurrent `Degraded` or `Down` (0.921 versus 0.701). For inventory `Understock`, combining inputs improves the score from 0.933 to 0.951. The time-to-event regression results are also task dependent. Using mean absolute error, the average absolute difference between predicted and observed durations, state features give the lowest error for inventory return to `Normal`; raw attributes give the lowest errors for manufacturing time to `Down` and time to recovery. The inventory target is named "Time to stable Normal recovery" in the exported results, but its generator uses the first future `Normal` observation without a persistence requirement. These findings support testing feature combinations for each target and population, rather than assuming that more inputs always improve prediction.

Pattern retrieval and occurrence matching. Six reference patterns were planted in each scenario. Pattern extraction uses the behavior-log projection, excludes passive snapshot event types, and retains at most three events on each side of an inter-state boundary. Within the same intra-state or inter-state family, the evaluator selects the candidate maximizing 0.7 times activity-sequence similarity plus 0.3 times the Jaccard similarity of participating object-type sets; ties favor the higher-ranked candidate. This retrieval comparison does not test equality of the full graph signatures in Definition 18. *Top-10 recovery* asks whether that selected candidate

Table 13: Prediction on later observations, evaluated outside FLOWVAULT. Positive test cases are shown as count/total and percentage. AUPRC is the higher observed score of the two classifiers within each feature set; bold marks the highest score in a row. These descriptive comparisons do not establish statistically significant differences.

		Prediction score (AUPRC)		
Target and horizon	**Positive test cases (share of test set)**	**State features**	**Raw attributes**	**All features**
Inventory				
`Understock` within 7 days	87/240 (36.3%)	0.875	0.927	**0.938**
`Critical Understock` within 7 days	19/239 (7.9%)	0.340	0.558	**0.581**
Manufacturing				
`Down` within 24 hours	25/248 (10.1%)	0.253	0.252	**0.325**
`Down` within 4 hours	4/244 (1.6%)	**0.287**	0.082	0.261

appears among the first ten ranked candidates: none of the six references passes this test in either scenario. The selected matches nevertheless resemble the planted activity sequences, with mean sequence similarity of 0.728 for inventory and 0.617 for manufacturing. Similarity ranges from 0 to 1, where 1 means an identical activity sequence.[5]

This resemblance does not show that the correct occurrences were recovered. Exact occurrence matching is generally poor, and a similar-looking pattern may concern different objects or time intervals. The immediate limitation is therefore both retrieval and occurrence matching: the evaluator finds related sequences, but the ranked output does not reliably direct the analyst to the planted behavior.

Conformance precision and recall. Recall is the fraction of planted violations detected; precision is the fraction of reported violations that match the reference. The inventory rules find 9 of 10 violations (90.0% recall), but also report 1,021 false positives. Manufacturing finds 7 of 9 (77.8% recall), with 801 false positives. Precision is consequently about 0.87% in both scenarios: fewer than one in a hundred reported violations matches a planted violation.

Several narrowly scoped rules recover their reference violations without false positives, whereas broad state-response rules generate most of the excess reports. The results therefore point to a need for more precise policy scope, object-role constraints, response windows, and treatment of incomplete observations. High recall alone does not justify using these rules as operational alarms.

Causal sanity checks. For each simulated intervention pair, the external script subtracts the untreated outcome from the treated outcome and averages these differences. A negative effect means a reduction in the outcome. The recovered reductions include 19.40 recovery hours and 17.80 shortage units in inventory, and 6.25 `Down` hours and 14.6 percentage points in failure probability in manufacturing.

Every reported effect equals the corresponding stored simulator contrast, giving zero absolute error against that reference. This verifies the paired-effect calculation and reporting pipeline. It does not estimate an otherwise unknown causal effect from the exported OCEL: both potential outcomes are supplied by the simulator. Applying causal analysis to an organizational log would still require a justified identification strategy and assessment of confounding and other assumptions.

7.6 Robustness and computational performance

The robustness study probes sensitivity to recording errors and configuration choices; the performance study measures the cost of native-core operations. Their scopes differ. Robustness was run on deterministic validation fixtures containing 52 inventory events and 68 manufacturing events, with eight leading objects in each, rather than on the two paper-scale logs. These tests identify implementation sensitivities but cannot establish robustness at the scale or diversity of an organizational deployment.

The perturbation harness records eleven runs per scenario. Its severity score is the maximum of four quantities: absolute macro-F1 change, event-state occupancy total variation, absolute endpoint-regime NMI change, and one minus top-ten pattern overlap. Scores at most 0.05 are labeled stable, scores above 0.05 and at most 0.15 conditionally stable, and higher scores sensitive. The stored classifications are one stable, one conditionally stable, and nine sensitive inventory runs, and three stable, one conditionally stable, and seven sensitive manufacturing runs. A change in one of these four measures can determine the classification even when the other measures remain unchanged.

Missing observations and temporal alignment cause substantial changes in these fixtures. With inventory event-attribute masking at probability 0.05, macro-F1 falls by 0.536, event-state occupancy total variation reaches 0.596, edge-rank correlation is -0.311, and top-ten pattern overlap is 0.25.[6] Missing comparisons can fall through to later CASE branches if the completeness flag is absent or stale, so explicit missingness checks are essential. In manufacturing, delaying alarm timestamps by five minutes gives edge-rank correlation 0.559 and lowers endpoint-regime NMI by 0.086. Changing the window length lowers endpoint-regime NMI by 0.074 in inventory and 0.167 in manufacturing. These changes are relative to each fixture's own baseline, not to Table 12.

The perturbation implementation further limits interpretation. The manufacturing threshold-scale run attempts to replace numeric thresholds in a query that instead reads the precomputed `degraded_latched` flag; it therefore leaves that query unchanged and is not evidence of threshold robustness. Sensor scaling does

[5] Sequence similarity is $1 - d_{\mathrm{Lev}}/\max(m, n)$, where d_{Lev} counts the insertions, deletions, and substitutions needed to transform one activity sequence into the other, and m, n are their lengths; it is defined as 1 when both sequences are empty.

[6] State-occupancy change is $\frac{1}{2}\sum_s |p_s - q_s|$ over event-state proportions. Edge-rank correlation is Spearman's correlation over the union of the 20 most frequent state-conditioned activity pairs, with absent pairs assigned frequency zero. Pattern overlap is the Jaccard coefficient between the ten highest-support keys in each run; the robustness evaluator uses family/activity-sequence keys, not the complete graph signatures of Definition 18.

not regenerate the degradation flags, and the nominal four-hour telemetry-gap setting marks a contiguous fraction of snapshots incomplete rather than selecting an exact four-hour interval. The robustness predictor also retains reference state histories. Consequently, unchanged state or prediction scores in these runs do not demonstrate resilience of a complete measurement-to-decision pipeline. Paper-scale perturbations that recompute all dependent features remain necessary.

Table 14 reports native-core performance. The single-import evaluation command takes 3.67 seconds for inventory and 23.64 seconds for manufacturing. It covers import, expression enrichment, graph construction, transition indicators, and automatic-state computation; it excludes the separate pattern command and Python prediction, conformance, and scoring steps. The complete external analysis stage takes 49.01 and 45.27 seconds, respectively. Thus the native-command difference cannot be attributed to predictor fitting. In controlled 300,000-event profiles, each measured operation completes within 2.49 seconds and peak resident memory remains below 0.81 GB.[7] The recorded environment is x86_64 Linux 6.12.101 with glibc 2.41, 12 logical CPUs, approximately 31.0 GiB RAM, and Rust 1.96.0 in release mode. The CPU model was not recorded. These single-repetition measurements support the feasibility of the tested native operations; they do not measure browser interaction latency.

7.7 Cross-scenario hypothesis assessment

Table 15 maps the observed evidence back to the hypotheses in Table 9.

8 Discussion

8.1 What the experiments establish

The two experiments support the central representation claim of this paper: a *state calendar* can be layered on top of object-centric event data without discarding the events, incidences, and object relations from which it was derived. Exact agreement with the synthetic oracles supports consistent event-state assignments, transitions, and episodes in the tested event-driven, single-leading-object setting; it does not validate every case covered by the formal model. More importantly, the derived quantities are not restatements of activity frequency. In inventory, `Normal` has fewer state-labeled events than `Critical Understock` yet slightly more object-time. In manufacturing, the pathway from `Degraded` through `Down` and `Recovery` back to `Running` is visible as a sequence of operational conditions even though the underlying activities and sensor events remain heterogeneous. Both observations support the use of state as an *analytical coordinate system* for persistence, entry, exit, recurrence, and recovery, in line with the broader motivation of state- and lifecycle-based process analysis [Hompes and van der Aalst, 2018, van Eck et al., 2016].

It is equally important to state what exact oracle agreement does not establish. The state query and the reference trajectory are two materializations of semantics supplied by the same simulator, so their agreement is a strong software and data-transformation test rather than an *external validation* of the chosen state taxonomy. In an organizational setting, policy thresholds can be versioned incorrectly, sensor values can arrive late, object relations can be incomplete, and experts can disagree about the operational meaning of a boundary. The perfect expression-based scores are therefore a prerequisite for domain validation, not a substitute for it.

8.2 Potential complementarity of manual and automatically detected states

High SOM purity and low NMI describe different aspects of the result: many windows share their cell's dominant simulator regime, yet the complete partition agrees weakly with the reference regimes. These reference regimes are distinct from the manual operational taxonomy. The resampling results concern the existing partition; they do not establish whether retraining would reproduce it. This is compatible with the intended role of automatic state discovery: lifecycle windows encode activities, attributes, and object context, so a cluster may separate two mechanisms hidden inside one manual state or merge manual labels that behave alike. Judging the automatic

[7] Peak RSS (resident set size) is the largest measured amount of physical memory occupied by the process. Table values use GiB (2^{30} bytes); GB denotes 10^9 bytes.

Table 14: Computational performance. Paper-run time covers the single-import evaluation command. The 300k column reports the slowest of import, state query, transition KPI, pattern, and automatic-state assignment for the controlled scale profile.

Scenario	Paper events	Paper time	Paper peak RSS	300k max / peak RSS
Inventory	46,256	3.67 s	1.09 GiB	1.52 s / 0.71 GiB
Manufacturing	43,458	23.64 s	1.00 GiB	2.48 s / 0.75 GiB

Table 15: Assessment of the pre-specified cross-scenario hypotheses.

ID	Assessment	Evidence and limitation
H1	Suggestive only	Manual states match the synthetic oracle, and learned cells provide retrospective warnings in inventory. Incremental benefit over manual labels alone and stability under refitting are untested; regime agreement, label transfer, and manufacturing warning precision are limited.
H2	Descriptive support only	State-determined views expose transition chains and related activity sequences. Zero top-10 recovery and weak occurrence matching limit the evidence for recovering planted mechanisms. The retrieval score does not test full graph equivalence.
H3	Not isolated	Combined features improve some tasks and worsen others. Conditional mutual information concerns state, not object context, and no full-minus-context comparison isolates the added value of context.
H4	Supported within benchmark scope	All core operations on 300,000 events complete within 2.49 seconds with less than 0.81 GB peak RSS. Browser latency, concurrent users, and larger feature dimensions remain unmeasured.
H5	Limited fixture evidence	Missing attributes, alarm delays, and window choices affect the small validation fixtures. Several perturbations retain dependent state inputs, and threshold scaling leaves the manufacturing query unchanged. Paper-scale robustness remains untested.

abstraction solely by one-to-one agreement with a rule-derived label would therefore miss its purpose [Berti et al., 2026c]. This motivates testing *complementarity*: expressions encode an accepted taxonomy, and discovered states propose additional distinctions. The current predictor comparisons combine manual and SOM descriptors within the state-feature group, so they do not isolate the incremental value of learned states over manual labels.

At the same time, the manufacturing warning results show that complementarity is not an excuse for weak validation. A precision of 0.097 and 2.48 false entries per machine-week would impose an unacceptable review burden, and low transfer balanced accuracy indicates that cell semantics are *population dependent*. The responsible workflow is therefore hybrid: use domain expressions wherever an accepted operational definition exists; use automatic states to generate hypotheses, identify sub-regimes, and inspect boundaries; and only then name, merge, split, or reject cells on explicit evidence. Model version, feature definition, training population, occupancy, and uncertainty should remain attached to every learned state assignment as part of its *provenance*.

8.3 Analytical utility is task dependent

The assessment separates two claims: explicit state semantics are needed to define the reported state-based results, while the benefit of state-derived predictor features is empirical and task dependent. Without the state calendar, the exposure, first-return, recurrence, and stuck-object findings have no operationally defined boundaries. The graph and prediction results then give quantitative evidence that state contributes information beyond activity order. Conditional mutual information remains positive in both scenarios, and state-only features are strong on several undesirable-transition tasks. At the four-hour manufacturing horizon, state-feature AUPRC exceeds raw-attribute AUPRC. This suggests that a *temporally persistent abstraction* can concentrate predictive information, but the comparison has only four positive test cases and uses a precomputed SOM representation. It is an exploratory result that requires a larger test and a pipeline fitted entirely within the training data.

The ablations, however, reject any universal claim that more context is always better. Raw features outperform the full representation on several group-holdout tasks, while the full representation helps on others. At least three mechanisms can explain this split. State features can compress noisy measurements and thereby improve temporal generalization. Raw measurements can retain *severity gradients* that a discrete state discards. Object-context counts can encode cohort-specific regularities that help within a period but transfer poorly to held-out groups. These are possible explanations, not isolated effects established by the current feature-family comparisons. SA-OCPM should therefore expose state, raw evidence, and object context as *separable feature families* and require ablations instead of assuming that their concatenation is optimal.

The pattern and conformance results matter for a different reason: they locate where the present implementation falls short. State-determined pattern views are visually interpretable and recover related control sequences, but the injected families do not appear in the top-k ranking and occurrence matching is weak. This points to sensitivity to graph canonicalization, pattern radius, support dilution, and the difference between structural similarity and exact instance identity. Likewise, high conformance recall combined with extremely low precision shows that broad post-state obligations are under-scoped. Future rule languages should make object roles, competing events, grace periods, censoring, and evidence of non-compliance explicit. Until then, pattern and conformance outputs are *diagnostic candidates* for analyst review, not autonomous alarms.

8.4 `Unknown`, robustness, and temporal semantics

The `Unknown` label is not a nuisance that can simply be filtered away; it marks the boundary of what the log can support. In both scenarios, frequent transitions through `Unknown` are driven by incomplete observations, and omitting them would fabricate direct transitions and bias dwell times. The small-fixture robustness experiment illustrates this point: missing attributes strongly affect inventory labels, and delayed alarms shift manufacturing edge ranks and learned-state agreement. Trustworthy state-aware analysis therefore depends on explicit *data-completeness semantics* and on provenance that records which evidence was available at each timestamp.

Short episodes and rapid switching require similar care. A threshold-based state notion can *chatter* when measurements oscillate near a boundary, and event-time materialization can create one-second `Setup` or transition episodes that are semantically valid yet unsuitable for some analyses. Minimum-duration filters, hysteresis, interval-valued change times, and a separation between physical and observational states are possible remedies. All of them should be configurable and reported, because smoothing removes noise but can also erase genuine short interventions or safety events.

Within their scope, the computational results are encouraging. Core operations stay below 2.49 seconds at 300,000 events on the tested native release build, with memory below 0.81 GB, which supports interactive back-end computation for moderate logs. The benchmark does not yet cover concurrent users, browser rendering of dense graphs, very high-dimensional windows, or distributed logs. The paper-scale manufacturing native command takes longer than its inventory counterpart despite a similar event count. The complete external analysis stages have the opposite ordering. Event count alone therefore does not explain cost, and the present timings do not isolate the contribution of each component.

8.5 Threats to validity and limitations

Several limitations constrain the interpretation of the assessment. First, both logs analyzed in the present experiments are synthetic. Their advantage is complete, versioned truth for states, transitions, injected patterns, conformance violations, interventions, and future outcomes. Their disadvantage is that the generators may share assumptions with the framework and cannot reproduce all recording errors, organizational workarounds, or semantic disagreements found in practice. The real-life applications summarized in Section 6.5 support practical feasibility in retail inventory, but do not establish the external validity of the complete integrated framework or its predictive, pattern-ranking, and conformance performance across domains. Those claims require further validation on independently collected inventory and manufacturing OCELs.

Second, the evaluation does not include human participants. No comparison measured whether analysts using SA-OCPM are more accurate, faster, or better calibrated than analysts using an ordinary OCDFG or lifecycle view, and expert interpretation of SOM cells and patterns was represented by oracle-based diagnostics. The *interpretability* of learned states is therefore only indirectly assessed, and analyst-facing utility remains an open question for future studies.

Third, several statistical results have limited precision. The main generators use one seed, the SOM bootstrap rescores a fixed map in three repetitions on a diagnostic sample, and each technical benchmark profile has one measured repetition. The prediction experiments also use SOM assignments and simulator-based regime mappings computed before the holdout splits, so they do not validate the complete discovery pipeline on unseen data. Some prediction holdouts contain very few positive examples. None of the three time-to-event regression tasks produced temporal-holdout scores because horizon purging left an empty split; the reported regression comparisons therefore concern only group holdouts. The displayed classification values select the better test score of two models and are therefore exploratory. Robustness was assessed only on small fixtures, and several perturbations do not recompute the dependent state flags or reference-derived predictor histories. The reported AUPRC values should be complemented by cluster-aware confidence intervals, repeated seeds, calibration plots, and prospective temporal validation.

Fourth, the 300,000-event scale profiles use controlled, low-dimensional features and do not reproduce the full 51- and 60-feature paper workloads, and the headless native measurements exclude browser layout and interaction latency. The scalability conclusion is therefore limited to the tested operations and hardware. Finally, the paired randomized causal experiment validates outcome bookkeeping under simulator-supplied counterfactuals. It does not address *causal identification* from observational OCELs, which still requires assumptions about confounding, interference among related objects, treatment timing, and measurement [Berti et al., 2026a, Qafari and van der Aalst, 2020].

8.6 Implications for SA-OCPM research and practice

The assessment suggests a *staged adoption path*. Organizations can begin with expression-based states whose semantics are already used in operations, validate the resulting calendars against independent records, and use transition, dwell, recurrence, and stuck-state views for diagnosis. Automatic state discovery can then search for sub-regimes and precursors, but learned cells should enter decision support only after criteria for stability, transfer, false-alert burden, and expert review are met. Prediction pipelines should retain ablations among raw, state, process, and object-context features, and pattern and conformance outputs should carry *evidence trails* and uncertainty rather than only ranks or binary alarms.

Methodologically, the next steps are to evaluate the framework on public and industrial OCEL 2.0 data, to conduct baseline-controlled analyst studies, to improve approximate and occurrence-level pattern matching, and to develop *probabilistic state calendars* that separate physical uncertainty from missing evidence. Transfer-aware SOM alignment, online drift detection, and hierarchical state notions could make learned states more portable. Conformance needs richer object-role and temporal semantics, and performance needs repeated browser and native benchmarks that vary relation density, feature dimension, lifecycle length, state count, and pattern diversity. These extensions will show whether the advantages observed in the synthetic scenarios translate into durable improvements in process analysis and decision making.

9 Conclusion

This paper presented State-Aware Object-Centric Process Mining, an end-to-end framework that makes the operational condition of objects explicit while preserving the multi-object evidence of an OCEL. Starting from object-centric event logs and directly-follows graphs, the formalization defines leading-object-specific, versioned state notions and derives state-aware event incidences, state changes, episodes, state-determined segments, state-aware graphs, and recurring intra-state and inter-state patterns. Because expression-based and automatically detected states share the same downstream interface, descriptive, performance, conformance, diagnostic, predictive, and decision-support analyses can all be organized around persistence and change without depending on how the states were obtained.

The framework is implemented in the FLOWVAULT browser workbench and evaluated in two reproducible, oracle-backed synthetic experiments. Expression-based states and transitions are materialized exactly against independent synthetic references, and the resulting calendars expose dwell, recovery, recurrence, and transition structure that traditional views cannot show, in both inventory management and manufacturing. State-conditioned control flow carries information beyond activity order, exploratory prediction comparisons favor state features on selected tasks, and controlled 300,000-event benchmarks show that the core back-end operations are computationally practical on the tested hardware.

The experiments also delineate the current boundary of the contribution. Automatically discovered states show high within-cell regime purity, but overall regime agreement and transfer of cell labels are limited, and stability after refitting on resampled objects remains untested. Their warnings suggest exploratory inventory prioritization, while manufacturing warnings impose a substantial false-alarm burden. Raw measurements remain competitive with state features, the combined feature set is not uniformly beneficial under group holdout, and the added value of object context is not isolated. Injected-pattern ranking is weak, and broad conformance rules produce too many false positives. Missing evidence and temporal alignment materially affect downstream findings. These limitations are not peripheral details; they determine when a state-aware result can support a decision and when it should remain an analyst hypothesis.

The overall conclusion is deliberately balanced. SA-OCPM provides a coherent representation and a useful analytical coordinate system for object-centric processes, with strong support for auditable expression-based states and promising but conditional support for learned states and predictive use. Building on the real-life applications reviewed in Section 6.5, establishing the decision value of the integrated framework requires further organizational studies, repeated and uncertainty-aware experiments, expert interpretation, and baseline-controlled user studies. The framework, its implementation, the evaluation artifacts, and the identified failure modes provide a concrete foundation for that next stage.

10 Statements and Declarations

10.1 Author contributions

Alessandro Berti: Software, implementation support, methodology, and writing - review and editing. Dina Kretzschmann: Conceptualization, ideation of SA-OCPM, methodology, and writing - original draft. Wil M. P. van der Aalst: Supervision, feedback, guidance, and writing - review and editing.

10.2 Funding

Funded by the European Union. This work has received funding from the European High Performance Computing Joint Undertaking (JU) and from the German Federal Ministry of Research, Technology and Space (BMFTR), the Ministry of Culture and Science of North Rhine-Westphalia (MKW NRW), and the Hessian Ministry of Science and Research, Arts and Culture (HMWK) under grant agreement No. 101250682.

10.3 Competing interests

Wil M. P. van der Aalst is a member of the editorial board of *Process Science*.

10.4 Data availability

The two observed synthetic OCEL 2.0 logs are publicly archived on Zenodo at `https://doi.org/10.5281/zenodo.22140119` [Berti, 2026b]. The complete evaluation bundle, including reference trajectories, injected-pattern and conformance truth, perturbation results, prediction outputs, causal validation records, and performance profiles, is available under `results/saocpm_15x/` in the fixed FLOWVAULT repository snapshot cited by Berti [2026a]. The Zenodo deposit contains the observed logs; the additional evaluation outputs are in that repository snapshot. Machine-readable summary tables and screenshot provenance are also included in the accompanying manuscript package under `evaluation-results/`.

10.5 Code availability

FLOWVAULT and the evaluation scripts are publicly available in the software repository [Berti, 2026c]. The generation and analysis code is identified by commit `b16000afd6a5271a6f822c5bd8118a1a01fcc74e`. The result bundle and four result screenshots are preserved in snapshot `fd86060c6113cacfc8efcfb0fa6c0e90dae7f79f` [Berti, 2026a]. The snapshot includes run manifests, configurations, dependency lockfiles, build instructions, and benchmark environment metadata. Git LFS is required to retrieve the full result files. These commit identifiers distinguish the tested code from later repository changes.

10.6 Ethics approval and consent to participate

Not applicable to the reported experiments because all evaluated event logs and reference outcomes are synthetic and no human participants were involved. Future studies involving organizational, personal, clinical, or human-participant data must obtain the approvals and agreements required for their setting.

10.7 Consent for publication

Not applicable.

References

Jan Niklas Adams, Daniel Schuster, Seth Schmitz, Günther Schuh, and Wil M. P. van der Aalst. Defining cases and variants for object-centric event data. In *2022 4th International Conference on Process Mining (ICPM)*, pages 128–135. IEEE, 2022. https://doi.org/10.1109/ICPM57379.2022.9980730.

Jan Niklas Adams, Emilie Hastrup-Kiil, Gyunam Park, and Wil M. P. van der Aalst. Super variants. In Andrea Marrella, Manuel Resinas, Mieke Jans, and Michael Rosemann, editors, *Business Process Management*, volume 14940 of *Lecture Notes in Computer Science*, pages 111–128. Springer, Cham, 2024. https://doi.org/10.1007/978-3-031-70396-6_7.

Alessandro Berti. FLOWVAULT: Synthetic evaluation results and screenshots. Software and data repository snapshot, 2026a. URL https://github.com/fit-alessandro-berti/flowvault/tree/fd86060c6113cacfc8efcfb0fa6c0e90dae7f79f/results/saocpm_15x. Commit fd86060c6113cacfc8efcfb0fa6c0e90dae7f79f. Accessed 14 September 2026.

Alessandro Berti. State-aware object-centric process mining: Two synthetic OCEL 2.0 evaluation logs. Zenodo dataset, version 1, 2026b. URL https://doi.org/10.5281/zenodo.22140119.

Alessandro Berti. FLOWVAULT. Software repository, 2026c. URL https://github.com/fit-alessandro-berti/flowvault. Accessed 14 September 2026.

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

Alessandro Berti, Johannes Herforth, Mahnaz Sadat Qafari, and Wil M. P. van der Aalst. Graph-based feature extraction on object-centric event logs. *International Journal of Data Science and Analytics*, 18(2):139–155, 2024a. https://doi.org/10.1007/s41060-023-00428-2.

Alessandro Berti, István Koren, Jan Niklas Adams, Gyunam Park, Benedikt Knopp, Nina Graves, Majid Rafiei, Lukas Liß, Leah Tacke genannt Unterberg, Yisong Zhang, Christopher Schwanen, Marco Pegoraro, and Wil M. P. van der Aalst. OCEL (object-centric event log) 2.0 specification, 2024b. URL https://doi.org/10.48550/arXiv.2403.01975. arXiv preprint arXiv:2403.01975.

Alessandro Berti, Dina Kretzschmann, and Wil M. P. van der Aalst. From object-centric event data to causal inventory insights: A structural equation model of understock and overstock, 2026a. URL https://doi.org/10.36227/techrxiv.177155632.23266403/v1. TechRxiv preprint.

Alessandro Berti, Dina Kretzschmann, and Wil M. P. van der Aalst. Segmentation for optimizing long-lifecycle processes in object-centric process mining. In Krzysztof Węcel, editor, *Business Information Systems Workshops*, volume 568 of *Lecture Notes in Business Information Processing*, pages 3–15. Springer, Cham, 2026b. https://doi.org/10.1007/978-3-032-11651-2_1.

Alessandro Berti, Dina Kretzschmann, and Wil M. P. van der Aalst. Interpretable execution state abstraction from object-centric event logs for process-aware decision support: Linking execution states to stock and policy regimes. In Roberto Posenato and Irene Vanderfeesten, editors, *Advanced Information Systems Engineering Workshops*, volume 586 of *Lecture Notes in Business Information Processing*, pages 321–333. Springer, Cham, 2026c. https://doi.org/10.1007/978-3-032-28160-9_29.

Massimiliano de Leoni and Wil M. P. van der Aalst. Data-aware process mining: Discovering decisions in processes using alignments. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, pages 1454–1461. ACM, 2013. https://doi.org/10.1145/2480362.2480633.

Anahita Farhang Ghahfarokhi, Gyunam Park, Alessandro Berti, and Wil M. P. van der Aalst. OCEL: A standard for object-centric event logs. In *New Trends in Database and Information Systems*, volume 1450 of *Communications in Computer and Information Science*, pages 169–175. Springer, Cham, 2021. https://doi.org/10.1007/978-3-030-85082-1_16.

Alexandre Goossens, Johannes De Smedt, Jan Vanthienen, and Wil M. P. van der Aalst. Enhancing data-awareness of object-centric event logs. In *Process Mining Workshops*, volume 468 of *Lecture Notes in Business Information Processing*, pages 18–30. Springer, Cham, 2023. https://doi.org/10.1007/978-3-031-27815-0_2.

Bart F. A. Hompes and Wil M. P. van der Aalst. Lifecycle-based process performance analysis. In *On the Move to Meaningful Internet Systems. OTM 2018 Conferences*, volume 11229 of *Lecture Notes in Computer Science*, pages 336–353. Springer, Cham, 2018. https://doi.org/10.1007/978-3-030-02610-3_19.

Ian T. Jolliffe and Jorge Cadima. Principal component analysis: A review and recent developments. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 374(2065):20150202, 2016. https://doi.org/10.1098/rsta.2015.0202.

Benedikt Knopp, Mahsa Pourbafrani, and Wil M. P. van der Aalst. Root cause analysis using rule mining on object-centric event logs. In *Process Mining Workshops*, volume 533 of *Lecture Notes in Business Information Processing*, pages 57–69. Springer, Cham, 2025. https://doi.org/10.1007/978-3-031-82225-4_5.

Teuvo Kohonen. The self-organizing map. *Proceedings of the IEEE*, 78(9):1464–1480, 1990. https://doi.org/10.1109/5.58325.

István Koren, Jan Niklas Adams, Alessandro Berti, and Wil M. P. van der Aalst. OCEL 2.0 resources – www.ocel-standard.org, 2024. URL https://doi.org/10.48550/arXiv.2403.01982. arXiv preprint arXiv:2403.01982.

Dina Kretzschmann, Alessandro Berti, and Wil M. P. van der Aalst. State matters: Detecting behavioral patterns in object-centric process mining, 2026a. URL https://doi.org/10.20944/preprints202607.1158.v1. Preprints.org preprint.

Dina Kretzschmann, Alessandro Berti, and Wil M. P. van der Aalst. State-aware object-centric process mining: Enhancing OCEL 2.0 with explicit state transitions. In *Enterprise Design, Operations, and Computing*, volume 16213 of *Lecture Notes in Computer Science*, pages 103–118. Springer, Cham, 2026b. https://doi.org/10.1007/978-3-032-15140-7_6.

Fabrizio Maria Maggi, Andrea Marrella, Fabio Patrizi, and Vasyl Skydanienko. Data-aware declarative process mining with SAT. *ACM Transactions on Intelligent Systems and Technology*, 14(4):1–26, 2023. https://doi.org/10.1145/3600106.

Najmeh Miri and Amin Jalali. Uncovering patterns in object-centric process mining: An approach using drill-down and roll-up techniques. In *Information Integration and Web Intelligence*, volume 15343 of *Lecture Notes in Computer Science*, pages 49–54. Springer, Cham, 2025. https://doi.org/10.1007/978-3-031-78093-6_4.

Gyunam Park, Jan Niklas Adams, and Wil M. P. van der Aalst. Conformance checking and performance analysis using object-centric directly-follows graphs. In *Business Process Management Forum*, volume 526 of *Lecture Notes in Business Information Processing*, pages 179–196. Springer, Cham, 2024. https://doi.org/10.1007/978-3-031-70418-5_11.

Viki Peeva, Marvin Porsil, and Wil M. P. van der Aalst. Object-centric local process models. In *Process Mining Workshops*, volume 533 of *Lecture Notes in Business Information Processing*, pages 376–388. Springer, Cham, 2025. https://doi.org/10.1007/978-3-031-82225-4_28.

Mahnaz Sadat Qafari and Wil M. P. van der Aalst. Root cause analysis in process mining using structural equation models. In *Business Process Management Workshops*, volume 397 of *Lecture Notes in Business Information Processing*, pages 155–167. Springer, Cham, 2020. https://doi.org/10.1007/978-3-030-66498-5_12.

Wil M. P. van der Aalst. Object-centric process mining: Dealing with divergence and convergence in event data. In *Software Engineering and Formal Methods*, volume 11724 of *Lecture Notes in Computer Science*, pages 3–25. Springer, Cham, 2019. https://doi.org/10.1007/978-3-030-30446-1_1.

Wil M. P. van der Aalst. Object-centric process mining: Unraveling the fabric of real processes. *Mathematics*, 11(12):2691, 2023. https://doi.org/10.3390/math11122691.

Wil M. P. van der Aalst and Alessandro Berti. Discovering object-centric Petri nets. *Fundamenta Informaticae*, 175(1–4):1–40, 2020. https://doi.org/10.3233/FI-2020-1946.

Maikel L. van Eck, Natalia Sidorova, and Wil M. P. van der Aalst. Discovering and exploring state-based models for multi-perspective processes. In *Business Process Management*, volume 9850 of *Lecture Notes in Computer Science*, pages 142–157. Springer, Cham, 2016. https://doi.org/10.1007/978-3-319-45348-4_9.