EPAComp: An Architectural Model for EPA Composition

Luís Henrique Neves Villaça*
luis.villaca@gmail.com
Federal University of the State of Rio de Janeiro (UniRio)
Rio de Janeiro, RJ, Brazil

Sean Wolfgand Matsui Siqueira
sean@uniriotec.br
Federal University of the State of Rio de Janeiro (UniRio)
Rio de Janeiro, RJ, Brazil

Leonardo Guerreiro Azevedo
lga@br.ibm.com
IBM Research
Rio de Janeiro, RJ, Brazil

ABSTRACT

Context: Complex Event Processing (CEP) architectures present high applicability in Realtime Streaming Analytics (RTSA) by extracting and generating valuable information from continuous data feeds, like in stock markets, traffic, and patient monitoring.

Problem: Although guidelines and models for CEP architectures have been proposed, the composition of its inter-operable elements in charge of processing events, known as Event Processing Agent (EPA), is challenging for software architects.

Solution: This work proposes EPAComp, a model that covers this gap and addresses large-scale processing requirements through features such as stream-based constructions and specialized EPAs.

IS Theory: We employed the Representation theory to create a model representing information systems for event processing.

Method: The model was applied in a real case experiment to create a solution to collect streams of events from around 200 systems and to provide a dashboard for monitoring their usage. Besides, industry experts qualitatively evaluated the proposal.

Results: The experiment results show an application of the model to handle heterogeneous data in a scalable and efficient manner according to indicators regarding performance, the assertiveness of processed output, degree of cohesion, and coupling of components. The qualitative results present that experts asserted EPAComp capabilities fit RTSA requirements.

Contributions: An architectural model for EPA composition that enhances the literature by (i) representing static and dynamic EPA compositions through arrangements of specific aggregation structures; (ii) defining the state-of-the-art event processing strategies in CEP; and, (iii) organizing the hierarchy of EPA types.

CCS CONCEPTS

• **Information systems** → **Computing platforms**; • **Computer systems organization** → **Real-time system specification**.

KEYWORDS

Complex Event Processing (CEP), CEP architecture, Event Processing Agent (EPA), Realtime Streaming Analytics (RTSA)

SBSI '23, May 29–June 01, 2023, Maceió, Brazil

ACM ISBN 979-8-4007-0759-9/23/05.
https://doi.org/10.1145/3592813.3592889

ACM Reference Format:
Luís Henrique Neves Villaça, Sean Wolfgand Matsui Siqueira, and Leonardo Guerreiro Azevedo. 2023. EPAComp: An Architectural Model for EPA Composition. In *XIX Brazilian Symposium on Information Systems (SBSI '23), May 29–June 01, 2023, Maceió, Brazil.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3592813.3592889

1 INTRODUCTION

Realtime Streaming Analytics (RTSA) scenarios receive one or more event streams as input and react to occurrences, often within a few milliseconds, producing one or more event streams as output [28].

Complex Event Processing (CEP) solutions extract and generate valuable information from continuous data feeds and have benefited from streaming technologies to handle RTSA scenarios, *e.g.*, stock markets, traffic, and patient monitoring [28, 35]. EPA (Event Processing Agent) processes events. CEP involves arrangements of interdependent EPAs, benefiting from each other capabilities [10].

EPA compositions express organizations that deal with a network of EPAs as just another EPA. Their design should address interoperability between EPAs and handle requirements such as efficiency and low coupling [10].

CEP solutions pose challenges to distributed applications with heterogeneous data to architects and system developers, like fault tolerance and complex integration [7]. EPA compositions bring requirements, like correlating occurrences according to a context [28], and drive matching services to business requirements [22].

The lack of standard models, semantics, and clear guidance on CEP studies makes the comparison of approaches hard [21]. The dispersed knowledge on this subject requires re-examining decisions through the research stages: planning and designing the artifact, implementing and evaluating a case, and documenting conclusions.

We investigated how to compose EPAs to meet RTSA demands. We propose EPAComp for it, bringing as benefits: explicit descriptions of CEP compositions, covering from simple aggregations to dynamic provisioning or decommissioning of EPAs; reuse of concepts; and, independence of underlying technologies.

We quantitatively assessed the proposal through an experimental study that collects user requests from around 200 systems. A complimentary evaluation presents the feedback of experienced system architects about the model's capability to meet RTSA requirements.

The remainder of the work is divided as follows. Section 2 and Section 3 presents the fundamental concepts and related work. Section 4 presents the EPAComp model, and Section 5 the model evaluations. Section 6 presents the conclusion and future work.

2 BACKGROUND

An event is an occurrence within a particular domain. A CEP architecture is composed of the following agents: *Producer*: input

events from external systems into CEP platforms; *Consumer*: pulls events from CEP platforms; and, *EPA*: processes events introduced by Producers or other EPAs and derive events by: *filtering*: acceptance or rejection of events; *transformation*: *e.g.*, enrichment of attributes, consolidation from multiple occurrences; and, *pattern detection*: inferences on, *e.g.*, time (*e.g.*, successive failures) or space (*e.g.*, recognition of objects in images) [10].

Those agents perform their tasks and interact with each other. Additional elements present in interactions are [10]:*Event Type*: represents event semantics and is used to guide the distribution of events among CEP agents; *Context*: a set of conditions used to correlate group events; *Global State*: refers to data from external sources to complement the event; *Channel*: provides CEP agents with event fetching and delivery capabilities.

Complexity rises when processing event groups in segregated contexts, *e.g.*, detecting which bank clients make more than 2 withdrawals within 24 hours requires (i) partition withdrawals events based on `client id` and `time` (24 hours since a withdrawal transaction), and (ii) aggregating events during specific time frames.

3 RELATED WORK

This section analysis conceptual models regarding: (i) main features; (ii) analytic patterns; and, (iii) industry demands.

Regarding **Conceptual Models main features**, Event Processing Technical Society (EPTS)[1] devise a reference architectural model [27] clarifying CEP design patterns based on commonalities from three CEP industry models:

- **IBM Conceptual Model** [24] classifies components according to their duties in: (1) Emitter: handles producer input data and delivers standardized events to the Bus; (2) Bus: receives events from Emitter in high frequencies and delegates processing to EPAs; (3) EPA: derives to the Bus a reduced amount of processed events; and, (4) Event Handler: deliver events from the Bus to consumers.
  Highlights: The similarities of Emitter and EPA (publishing via Bus) and Handler and EPA (consumption via Bus) foster reuse. Authors propose a "nested" architecture, where agents may contain a network of agents within themselves.
- **TIBCO BusinessEvents Model** [2] defines CEP tasks as: (i) Pre-processing: normalization and data cleansing on raw data; (ii) Event Tracking: identification and pre-selection; (iii) Situation Detection: relationships between events and historical data; (iv) Predictive Analysis: the impact of complex events on business processes; and, (v) Adaptive Business Processes: dynamic adjustment of processes.
  Highlights: Pre-selection tasks based on "events of interest" provide filtering mechanisms for EPA and Consumer. A distributed event-driven platform provides the communication infrastructure to enable high-performance event processing.
- **Oracle CEP Model** [25] correlates processing patterns: (i) Event sourcing: Producers publish events and trigger processing mechanisms from state transitions; (ii) Event schema: defines events from a tuple of attributes; (iii) Pattern matching: filters out redundant or irrelevant data, correlates meaningful events to infer critical decisions; and, (iv) Event stream: the base for processing, usually assisted by a time window for evaluation and correlation.
  Highlights: Authors propose an event sourcing strategy for the interaction among EPAs segments via segregated channels. Their model represents EPN based on *microservices*[2].

[1]EPTS develops standards and promotes understanding and advancement in the field of event processing.

Table 1: Realtime Streaming Analytics patterns [28]

1. Preprocessing - projecting (or filtering) from one data stream to the other.
2. Alerts and Thresholds - detecting conditions and generating alerts based on simple or complex conditions, such as the rate of increase.
3. Simple Counting, Counting within Segmented Windows - aggregating (*e.g.* Min, Max) in isolation or under a context (window).
4. Joining Event Streams - processing multiple data streams deriving a new event stream (using a pattern for joining operations).
5. Data Correlation, Missing Events, and Erroneous Data - matching events from different streams and detecting missing occurrences using redundant data to remove erroneous events from further processing.
6. Interacting with Databases - matching real-time streaming data against historical data sources.
7. Detecting Temporal Event Sequence - selecting temporal event sequences via regular expressions and acceptance criteria.
8. Tracking - tracking objects over space and time and detecting given conditions, *e.g.*, certify that cars adhere to speed limits.
9. Detecting Trends - detecting patterns from time series.
10. Running the Same Processing Mechanisms in Batch and Realtime Pipelines - combining batch and online processing.
11. Detecting and Switching to Detailed Analysis - detecting a condition that suggests an anomaly and further analyzing it using historical data.
12. Using a Model - training a model (often via Machine Learning) and then using it with the Realtime pipeline to make decisions.
13. Online Control - automatizing the resolution of problems like situation awareness and prediction of the next value.

According to EPTS, an EPA may be an individual node or an event processing network (EPN), *i.e.*, a collection of agents connected by channels [27]. However, EPTS does not provide details nor clear guidance to design EPA compositions.

Table 1 presents **RTSA patterns** proposed by Perera and Suhothayan for processing of continuous event flows [28].

Regarding **Industry Demands**, there are three aspects related to industry requirements to process large volumes of data in a short time: (i) Batch loads of events: online and offline processing of streams [5, 9, 19]; (ii) Incremental Model Training: *e.g.*, Machine Learning can improve pattern recognition accuracy during EPA processing [20]; and, (iii) Advanced filtering: moving filtering further into CEP pipelines reduces latency when constraints prevent doing it on event producers [12].

Table 2 consolidates the main concepts and existing CEP strategies proposed in the literature. Our model is an enhancement over existing proposals since it consolidates benefits from current studies and introduces relevant features for RTSA. Table 3 lists 17 requirements that our model meets compared to existing proposals.

Table 2: Main concepts and works that proposed them.

Concepts and strategies	References
Fundamental concepts and data structures	[2, 10, 24, 25, 28]
Segregation of responsibilities	[1, 11, 31, 34]
Handling high volumes of data	[8, 12, 29]
Batch processing of events	[5, 9, 19]
Stream processing on CEP	[3, 7, 16, 18, 22, 35]
Implementation of cases and metrics	[4, 6, 20, 33, 34]

[2]https://docs.oracle.com/en/solutions/learn-architect-microservice

Table 3: RTSA requirements handled by our model and their fit (*i.e.*, yes, no or partial) by other models.

Requirement	EPTS	IBM	TIBCO	ORACLE
R1 - EPA composition strategy	No	No	No	No
R2 - Batch loads of events	No	Yes	No	No
R3 - Advanced Filtering	No	No	No	Yes
R4 - Incr. Model Training	Yes	Yes	Yes	Yes
Requirement resulting from implementation of patterns presented in Table 1				
R5 - RTSA Pattern I	Yes	Yes	Yes	Yes
R6 - RTSA Pattern II	No	Partial*	Yes	Partial*
R7 - RTSA Pattern III	Yes	Yes	Yes	Yes
R8 - RTSA Pattern IV	Yes	Yes	Yes	Yes
R9 - RTSA Pattern V	Yes	Yes	Yes	Yes
R10 - RTSA Pattern VI	No	No	Yes	No
R11 - RTSA Pattern VII	Yes	Yes	Yes	Yes
R12 - RTSA Pattern VIII	Yes	Yes	Yes	Yes
R13 - RTSA Pattern IX	Yes	Yes	Yes	Yes
R14 - RTSA Pattern X	No	No	Yes	No
R15 - RTSA Pattern XI	No	No	No	Yes
R16 - RTSA Pattern XII	Yes	Yes	Yes	Yes
R17 - RTSA Pattern XIII	No	No	No	No

*trigger alerts based on simple threshold conditions [6, 28]

4 EPACOMP MODEL

This section presents the EPAComp model. We adopted the principles of Domain-Driven Design (DDD) [11] to correlate the CEP model components and classify them into cohesive units of analysis. DDD is a paradigm that solves problems associated with the composition of software components by focusing on core domain aspects and exploring models that emerge from functional (business) and non-functional (technical) requirements via a common language within bounded contexts.

Following the DDD principles, the EPAComp model was categorized into the following layers: (i) Domain: represents concepts and information about the business; (ii) User Interface: integrates with external actors; (iii) Application: defines the software-supported jobs; and, (iv) Infrastructure: supports higher layers via technical mechanisms. The most relevant entities are marked in grey. Some class diagrams are not presented due to a lack of space.

The model employs Java language constructs. We chose Java due to its relevance in industry - 52% of CEP commercial systems have been implemented using Java [7]. Although the Java use, the model can be adapted for any language with stream processing capabilities (like C#, C++, Scala, etc.) or one that integrates with streaming computation engines (*e.g.*, Spark, Storm, Flink, Kafka Streams). The notation is very similar across different languages.

4.1 Domain Layer

Represents business concepts such as entities and rules [11].

4.1.1 Event & Event Type (Figure 1). An `Event` has an `EventHeader` which points to `EventType` and `EventEmmitter`, *e.g.*, it is helpful for filtering. `EventOpenContent` allows a complementary free-format annotation that can help subscribers. `EventPayload` contains a set of `EventAttributes`, which extends `DataAttribute` by adding the indication if the data was *derived*. For high volumes of data, we can pull attributes mapped via `filterAttributes` (and `EventHeader` information), reducing traffic content. `EventFactory` is used by emitters to instantiate `Events` as they arrive in the CEP platform.

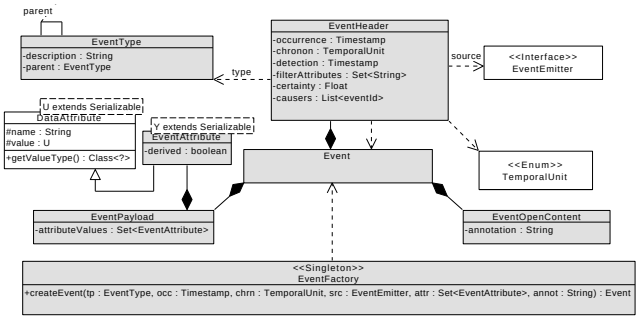

Figure 1: Classes of `Event` hierarchy.

4.1.2 Context (Figure 2). A `ContextComposite` is composed of one or more contexts. Contexts may be: `TemporalContext`, where every event starts a new window lasting a fixed amount of time comprising events that occur in a time frame; `FixedTemporalContext` is a `TemporalContext` with a fixed initial time (*e.g.*, `00:00` hours), and successive events within the same time range are assigned to the same Window and will not trigger a new window; `SegmentedContext` creates windows based on attribute values - each distinct combination of values corresponds to a different window; and, `EventContext` creates windows based on a list of event types representing the occurrence of a scenario of interest.

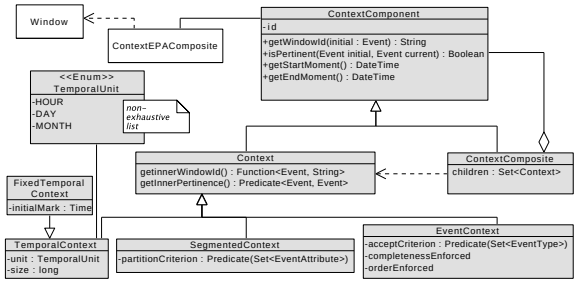

Figure 2: Classes of `Context` hierarchy.

4.1.3 Data Attribute. `DataAttribute` provides a data structure for *Serializable* attributes related to the events domain or technical drivers (*e.g.*, a timeout trigger).

4.2 User Interface Layer

This layer provides integration with external actors, which might sometimes be other computer systems [11].

4.2.1 Event Producer. The `EventProducer` injects events into the CEP Platform through publishing operations according to the `EventEmitter`. The `EventProducer` validates output events based on a Set of `EventTypes`, indicated on `PublishPattern`.

4.2.2 Event Consumer. `EventConsumer` pulls events from the CEP platform via `EventFetcher` interface implementation (*subscribeEvents*) by inspecting events according to `SubscriptionPattern`. It enables simple filtering and projection operations. It minimizes latency in data traffic [12].

4.3 Application Layer

This layer orchestrates the tasks the software should support, leveraging business domain objects [11].

4.3.1 Event Processing Agent. `EPAs` monitor events to detect patterns of information and trigger actions that output events. `EPAComponent` defines the common behavior of EPAs. It establishes a Channel and a StreamConsumer (for processing), fetches events based on SubscriptionPattern, and delivers them according to PublishPattern. `EPAComposite` aggregates `EPAs` into a set of components and delegates processing to them. Segregated inner channels perform operations within EPA compositions.

4.3.2 Stateless EPA (Figure 3). `StatelessEPA` processes input events independently of any other events (*i.e.*, no state is maintained).

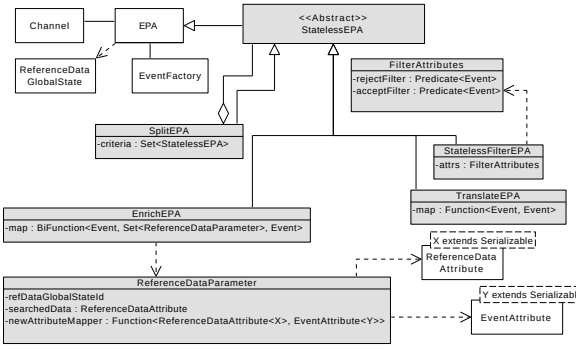

Figure 3: Stateless EPA classes.

`StatelessFilterEPA` filters events based on a criteria (from `FilterAttribute`) for their acceptance or rejection.

`TranslateEPA` receives a single event and generates another event.

`EnrichEPA` takes an event, pulls related information from a Global State, and derives an event with new attributes according to *map*, if triggered by PatternDetectEPA (Section 4.3.4). New attributes come from ReferenceDataGlobalState. It uses a ReferenceDataAttribute as a parameter to *newAttributeMapper*, to derive EventAttribute.

`SplitEPA` creates, from a single input event, a collection of events using multiple `StatelessEPA` instances, leveraging implementations from previous components.

4.3.3 Stateful EPA (Figure 4). `StatefulEPA` is influenced by previous events it has already processed [1, 10].

`StatefulFilterEPA`: its processing considers previous events (*e.g.*, fetch last `N` occurrences).

`AggregateEPA`: produces events as a function of an incoming stream of events (*e.g.*, average). The function is computed incrementally via a `reduceOperator` (BinaryOperator) along with an identity/initial value (`reduceOpIdentity`).

`ComposeEPA`: correlates events from two input streams. A subscription method is used for matching events according to an acceptance BiPredicate. A `map` BiFunction outputs derived events from matched pairs of source events.

4.3.4 Pattern Detect EPA (Figure 5). Makes event inferences.

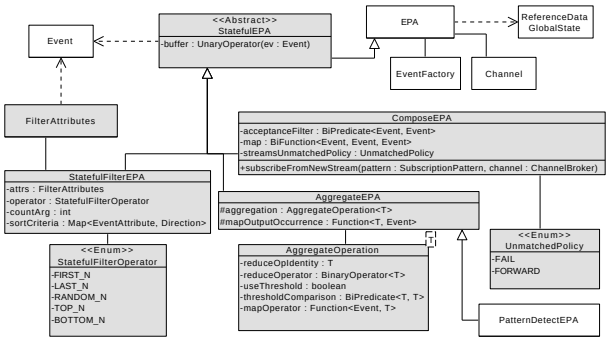

Figure 4: Stateful EPA classes.

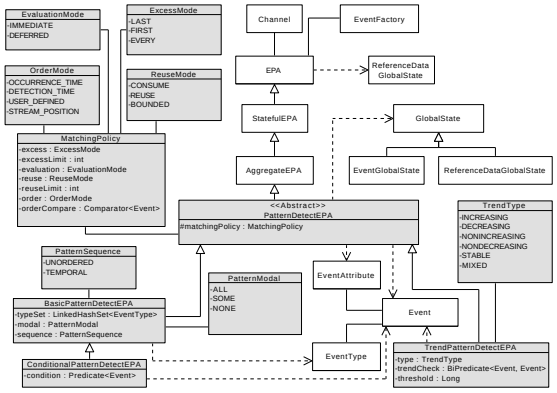

Figure 5: Pattern Detect EPA classes.

`BasicPatternDetectEPA` detects events that match a set of EventTypes. `PatternModal` indicates possibilities like: all event types should be met (ALL); it can be partially met (SOME) or not met (NONE).

`ConditionalPatternDetectEPA` requires a *condition* to be satisfied by matching events, defined via a Predicate.

`TrendPatternDetectEPA` detects trends by correlating pairs of subsequent events via *trendCheck* BiPredicate. Absolute or relative values can be evaluated (*e.g.*, target distance decreasing at a rate). Specific conditions may trigger events that further launch EnrichEPA, or dispatch a process via GlobalState.

`PatternDetectEPA` include a `MatchingPolicy` that indicates how to deal with events satisfying an expected pattern, like: (i) `EvaluationMode`: determines if the output is generated incrementally or at the end of a context; (ii) `ExcessMode`: when several instances of the same event type occur, consider the first, the last, or every one; (iii) `ReuseMode`: events are to be discarded after consumption or considered in subsequent analyses; (iv) `OrderMode`: a comparison parameter for event ordering (*e.g.*, occurrence or detection time).

These agents are usually processed under TemporalContext, where statistics are continuously provided.

4.3.5 Clustering EPA. `ClusteringEPA` continuously groups related events into a predefined number of clusters based on the events' features. A template parameter applies to event features (*i.e.*, one or more dimensional attributes). A *timeToLive* attribute balances

the relative importance of new data versus historical data, allowing faster reaction to changes. Inferred results may trigger further processing, *e.g.*, via EnrichEPA or via GlobalState.

4.3.6 Context Partitioner (Figure 6). Continuous processing of events associated with a context is performed according to segmented event grouping via `Window` and is handled by `ContextEPAComposite`. Dynamic provisioning and decommissioning of this agent happen according to the pertinence of events to `Windows`. When used along with `Context` to provision event grouping.

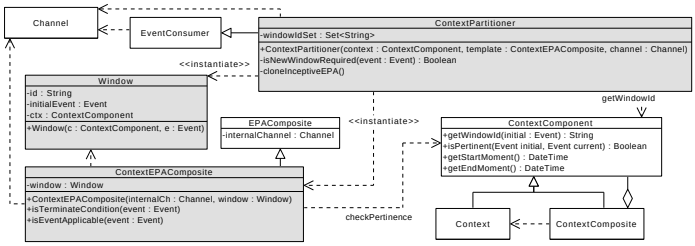

Figure 6: Context Partition classes.

4.3.7 Offline Event Loader (Figure 7). `OfflineEventLoader` leverages `EventProducer` to introduce batches of events from source systems into CEP platforms. It contains an instance of EventDataGlobalState to fetch an accumulated list of events, further sent to a CEP platform.

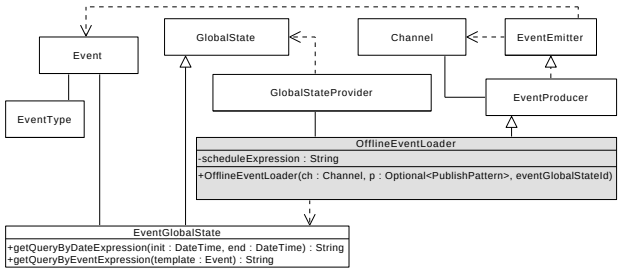

Figure 7: Offline Loader classes.

4.4 Infrastructure Layer

Provide technical mechanisms to support higher layers, such as interaction with event distribution platforms [11].

4.4.1 Channel (Figure 8). `Channel` instances capture events from event emitters, place them into the CEP platform via *publish* method, and route selected instances of events to event fetchers. SubscriptionPatterns assist filtering of fetched events.

Global Channel routes messages between CEP agents. For EPAs within an EPA composition, a segregated inner channel (one per composition) handles message exchange between them. `ChannelBroker` singleton provides both global and segregated Channel instances. Channels may optimize storage and reduce latency by serializing `Events` into `EncodedEvents` [17].

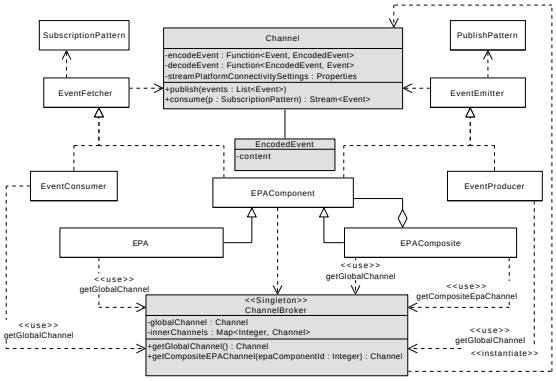

Figure 8: Channel classes.

4.4.2 Global State. `GlobalState` is used by EPAs to pull data outside the events' scope, bringing: (i) reference data, *e.g.*, to enrich data and drive processing logic, via `ReferenceDataGlobalState`; or, (ii) historical events, via `EventGlobalState`. It interacts with data stores via `GlobalStateDataStore`. Template parameters may be used to cast: `R`: a data-store-related record (*e.g.*, Row[3]); and `T`: a type associated to an `Event` entity or to a mapped `ReferenceDataAttribute`.

`AttributeContainers` properly identify data structures - *e.g.*, a *table*, under a *schema*, related via *parent* attribute.

EPAs supply a template (of `EventAttribute` or `ReferenceDataAttribute`) to derive a search expression for *query* based on their existing attributes. They use *findOne* to fetch a unique instance matching a template parameter.

To persist Events or Attributes, *updateMapper* Function from EventGlobalState or ReferenceDataGlobalState is used.

5 EVALUATION

This section presents the evaluation of our model, including the EPAComp coverage to mapped requisites, an industry case experiment, and feedback from experts.

Table 4 lists the requirements (Table 3) and the components that support each one. The model fully covers the requirements.

5.1 Industry Real Case Experiment

We evaluated the model in a real scenario employing a monitoring dashboard built using our model to investigate the benefits of EPAComp. The solution was evaluated in terms of effectiveness, given the demand and computational processes required and the interdependence degree among components.

5.1.1 Scenario Requirements. The scenario encompasses user interactions with 208 systems used by over 5000 employees and provides a dashboard of the utilization of the system aggregated by, *e.g.*, date ranges and user ids. The systems were grouped based on their strategy to capture events (Table 5).

5.1.2 Assessment Goals. We collected information on a week with two major transitions of non-monitored systems to monitored systems, indicating high loads of events during 2 nightly time-frames.

[3]https://spark.apache.org/docs/2.1.0/api/java/org/apache/spark/sql/Row.html

Table 4: Table presenting the components that support the requirements presented in Table 3.

Req.	Model Component(s)
R1	EPA, Channel, Context & subclasses, and ContextPartitioner
R2	OfflineEventLoader, GlobalState
R3	EventHeader, SubscriptionPattern
R4	PatternDetectEPA, ClusteringEPA
R5	EventHeader, SubscriptionPattern, PublishPattern, StatefulFilterEPA
R6	TranslateEPA, TrendPatternDetectEPA
R7	ContextPartitioner, AggregateEPA, Context-related classes
R8	ComposeEPA
R9	ComposeEPA and BasicPatternDetectEPA
R10	GlobalState & subclasses
R11	BasicPatternDetectEPA
R12	TrendPatternDetectEPA
R13	TrendPatternDetectEPA
R14	OfflineEventLoader, GlobalState & subclasses
R15	TrendPatternDetectEPA, ClusteringEPA, EnrichEPA
R16	ClusteringEPA
R17	TrendPatternDetectEPA, ClusteringEPA, GlobalState

Table 5: System groups

Group1 (G1)	Systems with no authentication mechanisms. Usage events are captured via web tracking;
Group2 (G2)	Systems with push notification. Systems that are already using publish mechanisms to send events into the messaging platform;
Group3 (G3)	Systems integrated into a Central Authentication Service (CAS). A job captures authentication events from a CAS repository;
Group4 (G4)	Legacy systems. A job captures login events from audit trails. A data source maps system user ids to CAS corporate ids.

Measurements included processing time and throughput (processed events per unit of time) indicators.

We evaluated correctness and completeness of results using the terminology [13]: true positive: events properly processed; true negative: events properly discarded; false positive: events incorrectly processed (improper rejection or with incorrect outcomes); false negative: events incorrectly discarded. Based on that, we calculated: accuracy: (true positives + true negatives) / total number of events; and, precision: true positives / (true positives + false positives).

5.1.3 Event Input and Output. Group 1 systems use web tracking, triggering *usage* events as web pages are loaded from a browser. Listeners for each system reject HTTP requests from invalid IPs (using a white list). Group 2 systems feed Events of types (*e.g.*, *login*, *logout*, *authentication error* or *usage*) into the CEP platform. Group 3 and Group 4 systems introduce batches of events into the CEP platform. For Group 3, an OfflineLoader periodically fetches usage events. Systems from group 4 capture usage events from audit trails, over distinct technologies, via OfflineLoaders.

5.1.4 Event Processing Activities. The processing of events is divided into five phases. In Phase I, the processing of raw events derives events representing user sessions: successive interactions with the system over time, based on login (if authentication applies) or IP addresses and timestamps (from anonymous requests). In Phase II, aggregations of user sessions are grouped by date. In Phase III, summaries of sessions are done on a date. In Phase IV, events coming from Phase III are clustered according to utilization summaries. In parallel, missing events on a date are detected, *i.e.*, it detects systems with no observed session. In Phase V, all previous events are processed in a dashboard that provides online monitoring and reporting with drill-down capabilities.

Table 6: Measurement metrics.

Δ**1** - Event Generation Delay - from the moment events are instantiated until the channel *publish* method is called.
Δ**2** - Channel Conversion Delay - within channel, from the moment event serializations start (*e.g.*, via Avro - https://avro.apache.org/) until a call to the messaging platform is performed.
Δ**3** - CEP EPA Processing Time - from the moment an event message is consumed by the EPA until either an event is derived by this component (usual StatelessEPA scenarios) or an event is processed and placed on a buffer (StatefulEPA scenarios).
Δ**4** - Fetcher Conversion Delay. Conversion from CEP Platform events to Dashboard API, which is fed with derived events.

Table 7: Processing time measurements.

	Latency (*milliseconds*)
Δ1 - Event Generation Delay	
Average Time Group 1, 2, 3, 4 Producers	0.1 - 0.5
Δ2 - Channel Conversion Delay	
Average Time Global / Segregated Channels	0.1
Δ3 - EPA Processing Time	
Average Time StatelessEPA	1 - 5
Average Time StatefulEPA	1 - 10
Average Time ClusteringEPA*	3000
Δ4 - Fetcher Conversion Delay	
Average Time Dashboard Consumer	0.1 - 0.5

*Clustering initialization takes around three seconds

5.2 Experiment Execution

The experiment was performed in four servers configured with an Intel Xeon processor (8 cores) and 32 GB RAM, running containers orchestrated via Docker Swarm[4]. Among Swarm features are scalability (guaranteed number of replicas) and load balancing (distributing containers between hosts).

Processing scenarios are regular loads (1000 events/day) and batch loads (up to 500.000 events per system at once). Batches are processed when applications are monitored when we load authentication data since January 2012 for historical analysis. Two overnight batches were performed, one with 15 systems and another with 24 systems.

Channel implementations were refactored to copy all published and subscribed events to new Topics (T_CH_PUB, T_CH_SUB). Code was injected in EPA to trigger the submission of messages to another Topic (named T_EPA). Every message contains processing start and end date-time and serialized representations of the output event.

5.2.1 Effectiveness. We evaluated T_EPA entries to: verify EPA processing time; correlate raw and derived events, confirming if all events from T_CH_PUB and T_CH_SUB were adequately processed assigned. Latency measurements were defined according to a benchmark study for CEP [21], based on metrics depicted in Table 6. Measurements results are presented in Table 7.

Results indicate that, on higher load occasions, throughput scaled up to around 100 raw events processed per second. We observed no compromise to integrity (all events were processed) nor any impact on availability, resulting in an average processing time (under

[4]https://docs.docker.com/engine/swarm/

Table 8: Coupling-Related Measurements

Class	CBO	NDO	RECBO
EventProducer	5	0	0
OfflineEventLoader	7	0	0
EventGlobalState	3	2 [i]	1 [i]
EventConsumer	5	0	0
ReferenceData-GlobalState	5	2 [i]	1 [i]
EnrichEPA	9	1 [ii]	0 [iii]
StatefulFilterEPA	7	1 [ii]	0 [iii]
ClusteringEPA	8	1 [ii]	0 [iii]
ContextEPA- Composite	9	1 [ii]	0 [iii]
AggregateEPA	8	1 [ii]	0 [iii]
Channel	5	$\propto$ [iv]	$\propto$ [v]

[i] GlobalStateProvider factory and CEP agents reference this instance at run-time.
[ii] At creation time: EPAComposite, ContextEPAComposite, or ContextPartitioner.
[iii] After creation, EPAs are not referenced by other model components.
[iv] Static Channel references scale based on the number of EPA specializations.
[v] Dynamic Channel references are proportional to the number of agents.

normal circumstances) of 10 ms. However, the total running time is impacted during batch loads.

All events were correctly assigned and processed by EPAs. There was no improper discarding of events (false negative) nor improper processing (false positive), and accuracy and precision achieved 100%. Around 550,000 raw events (historical events plus 10,000 contemporaneous events) and 350,000 derived events were observed.

5.2.2 Interdependence degree. We assessed our components' coupling and cohesion degree, which led to more reliable and maintainable products [14]. CEP agents interact with each other via the channel, presenting no knowledge of methods and attributes of each other (minimizing coupling among them). Measurements to corroborate this aspect were: *CBO* [26] (the number of times methods of a class use methods or attributes of another class); *NDO* [26] (the number of dependent classes of a class); and, *RECBO* [23] (the number of instances accessing the methods of a class at runtime). Table 8 presents those measurements. Observed CBO values (≤ 9), are indications of low coupling [30].

The LCOM metric [15] was used to evaluate the degree of cohesion of the Channel. CEP agents are dependent on Channel since they interact through it. It consists of the number of pairs of methods on disjoint sets of variables. Both exposed methods *publish* and *consume* of the Channel share *streamPlatformConnectivitySettings* (Section 4.4.1), so no disjoint set of edges arise from its graph - indicating LCOM = 1 (high cohesion [15]).

5.3 Feedback from Industry Experts

A complimentary evaluation of the capability of our model to meet the 13 RTSA pattern requirements (Table 1) was performed in a workshop with expert system architects. Participants were selected based on their background (Computer Science Graduation degree and experience developing solutions that handle high volumes of transactions near real-time). Five experienced professionals joined our workshop: two from the Finance area, one from Telecommunication, and two from Oil & Gas.

5.3.1 Sessions. The workshop was divided into 3 sessions.

In the first session, we presented the CEP components and the logic behind their compartmentalization into DDD layers. We also presented the executed experiment.

In the second session, we explained the evaluation requirements and RTSA patterns. Then, we asked the participants to fill out a questionnaire. They were all in the same room without the researcher during this activity.

In the third session, we debated the justifications for the answers, aiming to reach a consensus concerning the capacity of our model to meet the demands of RTSA patterns.

5.3.2 Workshop Questionnaire and Evaluation Criteria. For the first set of questions, we inquired participants if they understood the model, the requisites to be evaluated, the rules for evaluation, and whether they considered it reasonable.

We developed a plan to clarify our goals by following the GQM (Goals, Questions, and Metrics) strategy [32]. A questionnaire was presented with 1 question for each requirement of Table 3. Participants evaluated the adequacy of the model to each RTSA pattern by marking their perception as: *0*: does not meet the acceptance criteria; *1*: partially meets the acceptance criteria with severe restrictions; *2*: meets the acceptance criteria with minor restrictions; *3*: fully meets the acceptance criteria. They were asked to provide a textual justification for their answers.

5.3.3 Results. Positive feedback was given for all questions on the first four questions.

In the second set, we observed no participants assigned a grade of less than 2. The majority of the answers for this set received 3. Only 4 cases out of the 65 responses were evaluated with grade 2 (*i.e.*, it meets the acceptance criteria with minor restrictions).

The justified answers were further discussed among all participants and related to the following observations: on *Pattern 4*, correlations involving different data formats (audio, video) and attributes (as geolocation) were not directly approached and may require further extensions; on *Pattern 9*, an extension of PatternDetectEPA provides the prediction of clusters; however, additional extensions are required as we aim to incorporate functionalities such as supervised Machine Learning; and on *Pattern 13*, use cases that involve more complex online control scenarios involving inferences of corrective actions are needed.

We can infer potential benefits from the proposed model by observing the convergence in the answers from the five participants. Results indicate the model fully complies with 10 out of 13 patterns and meets acceptance criteria for the remaining ones - whereas, for EPTS Reference CEP Models (described in Section 3), only 9 out of 13 patterns apply [6, 28].

6 CONCLUSION

Designing of CEP architecture platforms aiming on inter-operable, reusable components requires clear understanding of EPA agents composition.

The main contribution of this work is the proposed architectural model which address weakness of literature with regard to EPA composition. The model provides a guidance to minimize complexity by establishing responsibility assignments for components, and by highlighting and isolating common functionalities.

The proposed model enhances current CEP literature by: (i) Representing static and dynamic EPA compositions through arrangements of specific aggregation structures; (ii) Representing state of the art event processing strategies in CEP, such as advanced filtering, offline introduction of events, incremental model training, and stream processing; (iii) Clarifying and reorganizing the hierarchy of EPA types, *e.g.*, by providing reuse of StatefulEPAs on SplitEPA, and by handling event processing via stream processing pipelines.

Compared to other architectural models, our proposal addresses solutions for patterns not yet covered, which is accomplished by the features like stream grouping based on context, processing of streams of events, and offline processing of event batches.

We executed experiments within an industry real use case which results demonstrate the model guides us to implement cohesive components that integrate events and agents through a platform (via message streaming) with a low degree of coupling. Other benefits perceived from this solution include: the ability to meet the demand for integration with systems that present constraints such as restricted schedules (via offline loads); advanced message filtering (via platform features); continuous model training; and, high performance and reusability for emitting and pulling events via streaming operations provided by specialized components. This analysis can serve as a guidance to elaborate other solutions, depending on the architecture application scenarios.

As limitations, the model encompasses 4 layers with more than 60 relevant concepts, which is a trade-off to express different composition scenarios. Besides, the evaluation results could have bias due to the researchers engagement in the implementation. We mitigate this by collecting feedback about the model by performing a workshop with expert system architects.

As future work, we point out the need for refining the model to encompass other relevant scenarios, *e.g.*, one in which events are inferred from data collect by sensor cameras geographically spread. Also, we plan to create guidelines to assist architects and developers in implementing following the proposed model.

REFERENCES

[1] Alexander Artikis, Opher Etzion, Zohar Feldman, and Fabiana Fournier. 2012. Event processing under uncertainty. In *Proceedings of the 6th ACM International Conference on Distributed Event-Based Systems (DEBS '12)*. Association for Computing Machinery, New York, NY, USA, 32–43.

[2] T. Bass. 2006. *Fraud detection and event processing for predictive business*. Technical Report. Tibco.

[3] Lars Baumgärtner, Christian Strack, Bastian Hoßbach, Marc Seidemann, Bernhard Seeger, and Bernd Freisleben. 2015. Complex event processing for reactive security monitoring in virtualized computer systems. In *Proceedings of the 9th ACM International Conference on Distributed Event-Based Systems (DEBS '15)*. Association for Computing Machinery, New York, NY, USA, 22–33.

[4] Mikel Canizo, Enrique Onieva, Angel Conde, Santiago Charramendieta, and Salvador Trujillo. 2017. Real-time predictive maintenance for wind turbines using Big Data frameworks. In *2017 IEEE International Conference on Prognostics and Health Management (ICPHM)*. IEEE, Dallas, TX, USA, 70–77.

[5] Paris Carbone, Asterios Katsifodimos, Stephan Ewen, Volker Markl, Seif Haridi, and Kostas Tzoumas. 2015. Apache flink: Stream and batch processing in a single engine. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering* 36, 4 (2015), 28–38.

[6] Gianpaolo Cugola and Alessandro Margara. 2012. Processing flows of information: From data stream to CEP. *Comput. Surveys* 44, 3 (2012), 15.

[7] Miyuru Dayarathna and Srinath Perera. 2018. Recent Advancements in Event Processing. *Comput. Surveys* 51, 2 (Feb. 2018), 33:1–33:36.

[8] Philippe Dobbelaere and Kyumars Sheykh Esmaili. 2017. Kafka versus RabbitMQ: A comparative study of two industry reference publish/subscribe implementations: Industry Paper. In *Proceedings of the 11th ACM International Conference on Distributed and Event-based Systems (DEBS '17)*. Association for Computing Machinery, New York, NY, USA, 227–238. https://doi.org/10.1145/3093742.3093908

[9] Godson Michael D'silva, Azharuddin Khan, Gaurav, and Siddhesh Bari. 2017. Real-time processing of IoT events with historic data using Apache Kafka and Apache Spark with dashing framework. In *2017 2nd IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology (RTEICT)*. IEEE, Bangalore, India, 1804–1809. https://doi.org/10.1109/RTEICT.2017.8256910

[10] Opher Etzion and Peter Niblett. 2010. *Event Processing in Action* (1st edition ed.). Manning, Greenwich, 74° w. long.

[11] Eric Evans. 2004. *Domain-driven design: tackling complexity in the heart of software*. Addison-Wesley Professional, Boston, USA.

[12] Eric Falk, Vijay K. Gurbani, and Radu State. 2017. Query-able Kafka: an agile data analytics pipeline for mobile wireless networks. *Proceedings of the VLDB Endowment* 10, 12 (Aug. 2017), 1646–1657. https://doi.org/10.14778/3137765.3137771

[13] Tom Fawcett. 2006. An introduction to ROC analysis. *Pattern recognition letters* 27, 8 (2006), 861–874.

[14] Norman Fenton and James Bieman. 2014. *Software Metrics: A Rigorous and Practical Approach, Third Edition* (3rd ed.). CRC Press, Inc., USA.

[15] Martin Hitz and Behzad Montazeri. 1995. Measuring coupling and cohesion in object-oriented systems. In *Proc. Int. Symposium on Applied Corporate Computing*. Monterrey, Mexico.

[16] Shweta Khare, Kyoungho An, Aniruddha Gokhale, Sumant Tambe, and Ashish Meena. 2015. Reactive stream processing for data-centric publish/subscribe. In *Proceedings of the 9th ACM International Conference on Distributed Event-Based Systems (DEBS '15)*. Association for Computing Machinery, New York, NY, USA, 234–245. https://doi.org/10.1145/2675743.2771880

[17] Kazuaki Maeda. 2012. Performance evaluation of object serialization libraries in XML, JSON and binary formats. In *2012 Second International Conference on Digital Information and Communication Technology and it's Applications (DICTAP)*. IEEE, Bangkok, Thailand, 177–182. https://doi.org/10.1109/DICTAP.2012.6215346

[18] Alessandro Margara and Guido Salvaneschi. 2013. Ways to react: Comparing reactive languages and complex event processing. *Workshop on Reactivity, Events and Modularity (REM 2013)* (2013), 14.

[19] Niels Martin, Andreas Solti, Jan Mendling, Benoît Depaire, and An Caris. 2021. Mining Batch Activation Rules from Event Logs. *IEEE Transactions on Services Computing* 14, 6 (Nov. 2021), 1908–1919. https://doi.org/10.1109/TSC.2019.2912163 Conference Name: IEEE Transactions on Services Computing.

[20] Christian Mayer, Ruben Mayer, and Majd Abdo. 2017. StreamLearner: Distributed Incremental Machine Learning on Event Streams: Grand Challenge. In *Proceedings of the 11th ACM International Conference on Distributed and Event-based Systems (DEBS '17)*. Association for Computing Machinery, New York, NY, USA, 298–303. https://doi.org/10.1145/3093742.3095103

[21] Marcelo R. N. Mendes, Pedro Bizarro, and Paulo Marques. 2008. A framework for performance evaluation of complex event processing systems. In *Proceedings of the second international conference on Distributed event-based systems (DEBS '08)*. Association for Computing Machinery, New York, NY, USA, 313–316. https://doi.org/10.1145/1385989.1386030

[22] John A Miller, Stephan Reiff-Marganiec, and Xiaofei Xu. 2019. Guest Editorial: Recent Advances in Web Services Research. *IEEE Transactions on Services Computing* 12, 3 (2019), 412–414.

[23] Áine Mitchell and James F Power. 2004. An Empirical Investigation into the Dimensions of Run-Time Coupling in Java Programs. In *Proceedings of the 3rd International Symposium on Principles and Practice of Programming in Java*. Trinity College Dublin, ACM, Las Vegas, Nevada, 9–14.

[24] Catherine Moxey, Mike Edwards, Opher Etzion, Mamdouh Ibrahim, Sreekanth Iyer, Hubert Lalanne, Mweene Monze, Marc Peters, Yuri Rabinovich, Guy Sharon, et al. 2010. A conceptual model for event processing systems. *IBM Redguide publication* 1, 1 (2010), 1–42.

[25] Oracle. 2010. Overview of Oracle CEP. https://docs.oracle.com/cd/E21764_01/doc.1111/e14476/overview.htm.

[26] Calvins Otieno, George Okeyo, and Stephen Kimani. 2015. Coupling measures for object oriented software systems-a state-of-the-art review. *Int. Journal Of Engineering And Science* 4 (2015), 01–10.

[27] Adrian Paschke and Paul Vincent. 2009. A reference architecture for Event Processing. In *Proceedings of the Third ACM International Conference on Distributed Event-Based Systems (DEBS '09)*. Association for Computing Machinery, New York, NY, USA, 1–4. https://doi.org/10.1145/1619258.1619291

[28] Srinath Perera and Sriskandarajah Suhothayan. 2015. Solution patterns for real-time streaming analytics. In *Proceedings of the 9th ACM International Conference on Distributed Event-Based Systems (DEBS '15)*. Association for Computing Machinery, New York, NY, USA, 247–255. https://doi.org/10.1145/2675743.2774214

[29] Medhabi Ray, Chuan Lei, and Elke A. Rundensteiner. 2016. Scalable Pattern Sharing on Event Streams. In *Proceedings of the 2016 International Conference on Management of Data (SIGMOD '16)*. Association for Computing Machinery, New York, NY, USA, 495–510. https://doi.org/10.1145/2882903.2882947

[30] Raed Shatnawi. 2010. A quantitative investigation of the acceptable risk levels of object-oriented metrics in open-source systems. *IEEE Transactions on software engineering* 36, 2 (2010), 216–225.

[31] Kia Teymourian and Adrian Paschke. 2010. Enabling knowledge-based complex event processing. In *Proceedings of the 2010 EDBT/ICDT Workshops (EDBT '10)*. Association for Computing Machinery, New York, NY, USA, 1–7. https://doi.org/10.1145/1754239.1754281

[32] Rini Van Solingen, Vic Basili, Gianluigi Caldiera, and H Dieter Rombach. 2002. Goal Question Metric Approach. *Encyclopedia of software engineering* (2002), 528–532.

[33] Carlos A. Velasco, Yehya Mohamad, and Philip Ackermann. 2016. Architecture of a Web of Things eHealth framework for the support of users with chronic diseases. In *Proceedings of the 7th International Conference on Software Development and Technologies for Enhancing Accessibility and Fighting Info-exclusion (DSAI 2016)*. Association for Computing Machinery, New York, NY, USA, 47–53. https://doi.org/10.1145/3019943.3019951

[34] Babak Yadranjiaghdam, Seyedfaraz Yasrobi, and Nasseh Tabrizi. 2017. Developing a Real-Time Data Analytics Framework for Twitter Streaming Data. In *2017 IEEE International Congress on Big Data (BigData Congress)*. IEEE, Honolulu, HI, USA, 329–336. https://doi.org/10.1109/BigDataCongress.2017.49

[35] Carlos Zimmerle and Kiev Gama. 2018. A web-based approach using reactive programming for complex event processing in internet of things applications. In *Proceedings of the 33rd Annual ACM Symposium on Applied Computing (SAC '18)*. Association for Computing Machinery, New York, NY, USA, 2167–2174.