# The Process-oriented Event Model (PoEM) – A Conceptual Model for Industrial Events

Om Prasad Patri
University of Southern California, Los Angeles, CA
patri@usc.edu

Vikrambhai S. Sorathia
Kensemble Tech Labs LLP
Gandhinagar, India
vsorathia@gmail.com

Anand V. Panangadan, Viktor K. Prasanna
University of Southern California, Los Angeles, CA
anandvp|prasanna@usc.edu

## ABSTRACT

The paper presents a comprehensive theoretical framework for modeling events and semantics of event processing at a level of abstraction that captures the different processes in industrial applications but is not limited to a specific application domain. The model, called Process-oriented Event Model (PoEM), provides a formal approach to model real-world entities and their interrelationships, and specifies the process of moving from data streams to event detection to event-based goal planning. The model links event detection to states, actions, and roles, enabling event notification, filtering, context awareness and escalation. PoEM defines event and non-event concepts and combines information from them to build an event processing workflow. Usage of the PoEM model is illustrated in case studies from the oil and gas industry and maritime piracy events.

## Categories and Subject Descriptors

H.2.1 [**Database Management**]: Logical Design—*Data Models*; I.2.4 [**Artificial Intelligence**]: Knowledge Representation Formalisms and Methods

## General Terms

Design, Theory

## Keywords

Data Streams, Complex Event Processing, Semantic Web, Event Ontology, Event Escalation

## 1. INTRODUCTION

Event stream processing and complex event processing (CEP) approaches have emerged as potential solutions for processing streaming data in a range of industrial applications [11]. A comprehensive CEP system designed for the enterprise would include event detection, filtering, notification, action determination, context awareness, visualization and escalation mechanisms [5]. Existing CEP systems are designed to perform continuous queries against the incoming data stream for detecting complex events [2, 6, 8]. These systems are configured to perform specific actions according to pre-defined business rules over the detected events. These CEP approaches are therefore focused on event detection and filtering without making provisions for the other processes, such as suggesting appropriate follow-up actions from background knowledge, or escalating high priority events to reduce response times.

Existing event processing approaches are based on a broad definition for events, classifying *anything which happens* as an "event" [5]. Such a broad scope makes it hard to apply the approach to a real-world industrial application since only a relatively small number of the possible events are important and should be processed further. We provide a precise mathematical definition for an "event" which connects events to real-world entities, their properties and timestamps, enabling us to retain only relevant observations. Moreover, industrial processes have to be described in detail using a formal ontology for event automation. A generic event ontology-based approach is thus not directly applicable to real-world applications.

A comprehensive model for event processing would enable complex event analysis across diverse underlying systems, people, entities, actions and happenings. The need for adopting a well-developed event model in enterprises has been stressed by Luckham and Schulte[1]. Recently, Etzion and von Halle[2] also motivated the issue of focusing on event modeling aspects rather than code and implementation aspects of event processing. In this paper, we propose a novel conceptual model called *PoEM* (Process-oriented Event Model) for complex event processing that attempts a comprehensive representation of processes seen in modern industries.

The PoEM model brings together, in a unified framework, the different types of entities that are expected to be present at different stages of an event-processing workflow and a formal specification of the relationships between these entities. Figure 1 shows the major components of PoEM (these components are described in detail in Section 3). PoEM has a detailed representation of concepts related to converting data streams to events (measurements, observations, interpretations) since most industrial events are detected from


*DEBS'14,* May 26 - 29, 2014, MUMBAI, India.

http://dx.doi.org/10.1145/2611286.2611291.

[1] http://www.complexevents.com/2013/01/23/why-companies-should-develop-event-models/
[2] http://www.slideshare.net/opher.etzion/er-2013-tutorial-modeling-the-event-driven-world

sensor-based data. Events are detected using pre-defined event profiles and this is used to drive goal planning. In PoEM, planning is performed using an explicit state model. The output of the planner defines the roles and actions that need to be taken to execute the plan. There is an event escalation component that serves to prioritize events, which is important in large industrial applications. Note that such a comprehensive event model can also form the basis for event-based integration of related data processing systems.

The contributions of this paper are 1) a comprehensive event model and ontology designed to describe the end-to-end process of moving from data to events to proactively responding to the events, 2) a step-by-step identification of key entities, observable properties, measurement models, interpretations, goal planning, actions, and roles that defines an event space applicable to practical industrial applications, 3) a state-based approach to goal planning based on detected events, and 4) description of the application of the PoEM approach to two real-world use cases.

The rest of the paper is organized as follows. A motivational use case of data management in an oil and gas enterprise is presented in Section 1.1. Section 2 describes related work. Section 3 describes the proposed model and the relevant foundational concepts, event concepts, processing concepts and the complete CEP workflow based on the model. We illustrate the application of the PoEM model to our motivational use-case in Section 4. We conclude in Section 5.

## 1.1 Use Case in the Oil and Gas Industry

We illustrate a potential use-case of the PoEM model for enterprise data integration in the oil and gas (O&G) industry. Due to high frequency of event-related information in the O&G industry [1] and criticality of quick responses for business as well as health, safety and environmental aspects, it is pertinent to use event processing approaches that can handle structured, semi-structured and unstructured information extracted from system silos. We have described how CEP-based approaches can offer faster response times, reduced data seeking efforts, efficient interaction patterns, and management by exception for the digital oilfield [15].

Petroleum production involves complex data-intensive processes that employ specialized tools and vendor products resulting in system silos, particularly across maintenance and production databases. For instance, a pump's working condition is monitored in operations systems. However, in the case of its failure, the responsibility of repairs is handed over to the maintenance team, who use separate tools for computerized maintenance management systems (CMMS). The pump failure event affects different teams such as operations, maintenance and production optimization and these teams are required to collaborate across system boundaries to bring the pump back to functioning status. The PoEM framework can be used to specify the life-cycle of such events (starting from the sensor level) and automate the generation of appropriate responses (such as initiating a repair ticket in the repair CMMS) and escalating the event priority if the expected goal is not reached.

# 2. RELATED WORK

Initial ideas on methods for interacting with event-based sense and respond systems are discussed by Chandy et al. [4] but no model for events or event processing is provided. Hinze [10] provides early work on identifying event profiles and constructing an event algebra for notification services. Event algebras are also proposed in related approaches by Zimmer and Unland [25], Eckert et al. [8] and Anicic et al. [2]. Events are typically modeled as data tuples consisting of attributes, values and timestamps, such as the 3-tuple by Voisard and Ziekow [21], the 2-tuple by Zhou et al. [24] or XML schema [3]. The E* event model [9] is useful for modeling multimedia events. A probabilistic event model for processing uncertain events is proposed by Wasserkrug et al. [22]. However, these approaches do not provide a model for describing an end-to-end workflow, moving from data streams to detecting and responding to events. Event processing has also been viewed as a process by the business process management (BPM) community [23] but such works typically focus on representing business logic. Cugola and Margara [6] provide a survey of several existing stream processing systems and models.

Several CEP systems use conceptual frameworks which are tightly coupled with a specific domain (such as CIDOC CRM[3]) and cannot be applied to other domains easily. On the contrary, generic models such as the Simple Event Model [20], Event Ontology[4], E* [9] or LODE [16] mainly provide a model for events and not the complete event processing workflow. They are not able to incorporate complex events, entities, actions, roles and inter-relationships between event and non-event concepts. Teymourian et al. [19] propose some of these concepts however, complex interrelationships between them are not sufficiently covered.

# 3. DESIGN OF THE PROCESS-ORIENTED EVENT MODEL (POEM)

Based on the event processing challenges discussed as well as requirements identified in industrial use cases, we enumerate key elements of the PoEM model. The first step is to model real world entities and their properties. This leads to modeling observations associated with entities and detecting events by interpretations of their instantaneous status. Events and event-related concepts are formally defined and linked to identification of relevant states, actions, and roles. A mechanism for event escalation is also proposed. The PoEM model can be rapidly adapted to handle complex business rules and domain-specific concepts, as illustrated in the case studies in Section 4.

## 3.1 Foundational Concepts

We start by modeling key elements in the universe of discourse (UoD). Beginning with entities of interest, the modeling procedure leads to identification of other related concepts, properties and interdependencies. The PoEM model adopts some foundational concepts from the dynamic information management methodology proposed by Sorathia [17]. This modeling approach, originally proposed for situational awareness, provides an effective framework to build a conceptual model.

### *3.1.1 Entity*

*Entities* are basic elements in the conceptual model. An entity may be physical or logical. We represent the set of entities in UoD as $K$ while individual entities are represented

[3] `http://cidoc.ics.forth.gr/OWL/cidoc_v4.2.owl`
[4] `http://purl.org/NET/c4dm/event.owl`

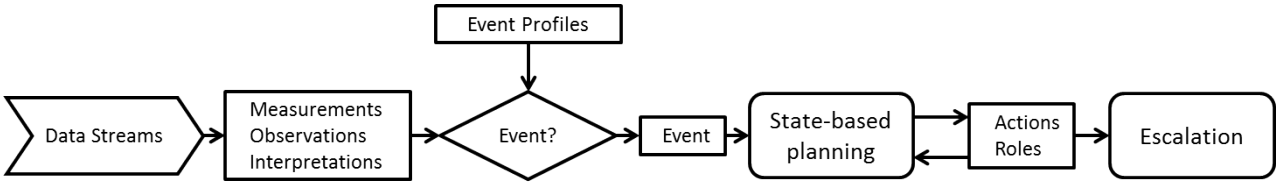


**Figure 1: Overview of components of the PoEM event modeling framework**

as $\kappa$. In the oilfield use case, entities would include physical entities such as pumps, wells, pipes, vehicles and human operators as well as logical entities such as databases and software applications.

$K = \{\kappa_1, \kappa_2, \kappa_3, \ldots\}$
Examples : pump-1, operator-A, vehicle-ZXV34

### 3.1.2 Observable Property

Each identified entity may have several *properties*. The next step is to identify and enumerate the set of relevant *observable properties*. The set of relevant observable properties is represented by $\Pi$, which includes individual members denoted by $\pi$. Observable properties may range from detecting the mere presence of an entity to various physical and chemical properties requiring sensing techniques.

$\Pi = \{\pi_1, \pi_2, \pi_3, \ldots\}$
Examples : temperature, pressure, motion, vibration

Observable properties can be grouped by the chemical, physical, and other scientific methods used for measurement, and they can be measured in different ways. To accurately capture properties associated with entities, we need a model for *measurement*.

### 3.1.3 Measurement

Identification of observable properties for the entities of interest leads to specification of how observations can be recorded in various ways. For instance, temperature is an observable property measured using digital and analog sensors (thermometers). Thermometers may provide readings in degrees Celsius or Fahrenheit. Also, sensitivity of the measuring device may vary significantly affecting precision and accuracy. Some sensors provide a continuous stream of readings, whereas others do so at intervals. As subtle changes in observable properties could be critical in high reliability operations, the modeling procedure must comprehensively handle these aspects for which we propose the concept of *measurement*. A set of measurements is denoted by $M$ while an individual measurement is shown as $\mu$. A measurement record captures the value of a sensor reading, along with its unit of measure (UoM), type and other details related to measurement. An instance of measurement can be indicated by the 3-tuple:

$$\mu_{S,\pi} = \langle \text{value, type, uom} \rangle$$

Here, $S$ is a specific sensor type (thermometer) and $\pi$ is the observable property (temperature) supported by the sensor. The same property can be measured using different sensors, each of which may have specific units of measure, sensitivity and operating ranges. This representation could include additional features related to precision, accuracy or sensor update frequency. However, once such information is identified for a specific sensor, it remains static. Therefore, a measurement model leads to a stream of readings stored as observations.

### 3.1.4 Observation

Measurement concepts enumerate all possible ways the observable properties can be measured. However, in order to interpret the current status, it is useful to retrieve instantaneous values from the sensor. This is achieved by introducing the concept of *observations* ($\Omega$), which is the set of all measurement observations. Individual observations ($\omega$) can be recorded at time $t$, for a given entity $\kappa$ as per a specific measurement model $\mu$.

$\Omega^{\kappa,\mu} = \{\omega_{t_1}^{\kappa,\mu}, \omega_{t_2}^{\kappa,\mu}, \omega_{t_3}^{\kappa,\mu}, \ldots\}$
Example : $\Omega^{k_1=pump,\mu_1=temp} = \{25, 26, 28 \ldots\}$

From the measurement model ($\mu_i$), property $\pi$, UoM and other relevant context can be determined. Sensors typically provide such values in the form of data streams. Digital oilfields are typically highly instrumented and provide many such sensor streams. These streams are often configured to feed into data historian systems.

### 3.1.5 Data Stream

Sensors reporting values as per specific observations ($\omega^{\kappa,\mu}$) result in continuous *streams* of data. We define $\Theta$ as the set of all data streams available for a given entity with individual instances of streams represented as $\theta$ as follows:

$\Theta = \{\theta_1, \theta_2, \theta_3, \ldots\}$
Examples : temperature stream, pressure stream

A specific data stream ($\theta$) is linked with observations ($\omega$) for a specific entity ($\kappa$) according to a specific measurement model ($\mu$) that determines the type of sensors and details regarding UoM and other relevant details.

### 3.1.6 Interpretation

Raw data values recorded and reported by sensors do not provide insights unless they are evaluated in a specific context. For instance, a thermometer reporting $25^{\circ}C$ for a tool room might be a standard condition, however the same value for a cold storage facility may be an exception. We introduce the concept of *interpretations* to resolve this issue. An interpretation provides meaning to the recorded values. Users can plug in their own event detection rules and filters here. For instance, for a given observation $\omega_t$, the recorded value may fall into a range that might be normal or critical. This directly provides contextual information about the entity. Therefore, it is useful to identify such ranges of values that makes them critical. Interpretation set (denoted by $X$ or $X^{\mu}$) is a set of all possible interpretation instances ($\chi$) that

can be identified for a specific measurement model ($\mu$) associated with an observable property.

$$X = \{\chi_1^\mu, \chi_2^\mu, \chi_3^\mu \ldots\}$$
$$\text{Examples} : \chi_1^{[a,b]}, \chi_2^{[c,d]}, \chi_3^{[e,f]}$$

Here, $\chi_1$ is the interpretation provided when the value of $\omega_{t_1}$ is in the interval $[a, b]$. Similarly, domain experts and practitioners can provide all the ranges and respective interpretations. An interpretation set for a sensor measuring pH of water may be represented as:

$$X_{\text{pH}} = \begin{cases} \text{Failed} & : \text{Sensor reading N/A} \\ \text{Invalid} & : (\text{pH} < 0) \cup (\text{pH} > 14) \\ \text{Acidic} & : 0 \leq \text{pH} < 7 \\ \text{Neutral} & : \text{pH} = 7 \\ \text{Basic} & : 7 < \text{pH} \geq 14 \end{cases} \quad (1)$$

Each interpretation ($\chi_i^\mu$) is associated with an evaluation condition, forming a branch ($i$) of the set ($X$). As seen from the above example, there may be a branch dedicated to find out whether the sensor is functioning and providing values in the first place. Placing this branch at the top of the interpretation set may lead to faster detection of failures, in case no data is being read. The evaluation condition is usually a function of the observation ($\omega$), and is denoted as $c(\omega)$. Then, an interpretation set ($X$) can be defined as:

$$X^\mu = \bigcup_{i=1}^{m} \{\chi_i^\mu \mid c_i \implies \chi_i^\mu\} \quad (2)$$

which, when expanded, leads to Equation 3.

$$X^\mu = \begin{cases} \chi_1^\mu & : \text{if } \text{c}_1(\omega) = \text{true} \\ \chi_2^\mu & : \text{if } \text{c}_2(\omega) = \text{true} \\ \ldots & \\ \ldots & \\ \chi_m^\mu & : \text{if } \text{c}_\text{m}(\omega) = \text{true} \end{cases} \quad (3)$$

For each measurement model $\mu$, there is at least one interpretation set delineating all possible interpretations which are related to values of the given property. The branches of the interpretation set are non-overlapping and only one of the branches is activated during evaluation. Individual interpretation sets can be defined based on piece-wise functions, boolean functions or other suitable mathematical or logical representations recommended by domain experts or practitioners.

So far, we have focused on a single observation value determined at a specific point. However, there can be additional interpretations derived for multiple observations as well. Considering readings at multiple time instances enables representation of complex scenarios such as a sudden increase or decrease in property measurements. For instance, temperature readings of $31°C$ at $t_1$ and $36°C$ at $t_2$ may be considered normal as they both belong to the normal range of $[30 - 37]°C$. However, two consecutive readings indicating an increase in excess of 10% may be a critical change according to a rule in a specific domain.

$$X_t^\mu = \chi^\mu \quad : \text{if } \Delta(\omega_{t_1}, \omega_{t_2}) \geq 10\% \quad (4)$$

Once all relevant interpretation sets are identified by domain experts and practitioners, they can also be utilized to determine the state of an entity. For instance, in case of the pH of water, given interpretation set and instantaneous observations, it may be possible to determine if water is in the "potable" state or not. Therefore, it is intuitive to determine the state of the entity based on various interpretation sets. We present the state model in Section 3.3.1.

## 3.2 Event Concepts

The foundational concepts proposed provide a mechanism to identify key entities, attributes, measurements, and interpretations. This lays the foundation for identification of events. The interpretations link the current measurement to identification of changing situations in the UoD. Therefore, interpretations directly link to identification of events and related concepts.

### 3.2.1 Event

In conventional event definitions, every change is declared to be an event. Therefore, as per our model, all recorded interpretations should result in events; however, this leads to an exponential number of events. For instance, even if temperature is recorded to be normal, any minor change in temperature will be reported as a new event. To address this issue, we introduced the concept of interpretation for specific ranges. In addition, changes in interpretation may not be relevant to the user's interest. For example, only extreme variations in temperature might be of interest. Taking these factors into account, we propose a new definition for an event that identifies a subset of interpretations to be considered as events. From all interpretations in an interpretation set, any one can be true at given point in time ($\tau$), which leads to identification of a *simple event* ($e_s$).

Definition 1. *An event is the interpretation of an observation of interest. Conceptually, a simple event is represented as*

$$e_s = \chi_{\omega^{\kappa,\pi,\tau}} \quad (5)$$

*where e refers to an event, $\tau$ is the time stamp of the event, $\kappa$ is the entity associated with the event, $\pi$ is the observable property associated with the entity for which this event was recorded, and $\chi$ is the chosen interpretation of the event.*

In the digital oilfield use case, a "high temperature" interpretation for a pump may lead to detection of a simple event as depicted in Figure 2. The figure gives a complete picture of the foundational concepts. The entity of interest, $\kappa_{pump}$, has observable property $\pi_{temp}$ which was observed using some measurement model. The observation $\omega_{t_1}$ was $100°F$, which, by referring to the interpretations for a pump leads to the identification of a simple event ($e_s$).

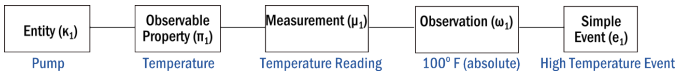


**Figure 2: Simple event identified in an oilfield**

### 3.2.2 Event Profile

As defined by Hinze [10], the concept of an *event profile* typically involves a query that is used to determine if a specific event has occurred or not. We can use interpretations and simple events to determine the event profile. Event profiles suggest what needs to be queried to determine the occurrence of an event.

For a simple event, the event profile consists of just an observation that can be performed once or repeated at specific time intervals. Equation 6 indicates a simple event profile that requires a single (one-time) observation of property $\pi$ for an entity $\kappa$ at time instance $\tau$.

$$P_s^o = \omega^{\kappa,\pi,\tau} \tag{6}$$

However, in real world scenarios, the requirements can be more complex. An event profile may constantly need to be evaluated at a specific time interval $\Delta\tau$. In such scenarios, a recurring event profile $P_s^r$ can be represented as in Equation 7. An example of an event profile to check the temperature of a pump every 30 seconds is also shown.

$$\begin{aligned} P_s^r &= \omega^{\kappa,\pi,\Delta\tau} \\ P_s^r &= \omega^{\text{pump,temp,30}sec} \end{aligned} \tag{7}$$

The outcome of an event profile query is a set of observations that can be evaluated based on the interpretation sets. For instance, suppose the value of pH of water was observed at a certain time to be 1.5. On evaluating this value based on the interpretation set provided in Equation 1, it is ascertained that the recorded value belongs to the "Acidic" branch in the interpretation set and all other branches are set to false as shown below. Hence, occurrence of a specific event related to the "Acidic" interpretation can be perceived.

$$\begin{cases} \text{False}: & \text{Failed} \\ \text{False}: & \text{Invalid} \\ \text{True}: & \text{Acidic} \\ \text{False}: & \text{Neutral} \\ \text{False}: & \text{Basic} \end{cases}$$

So far, we have only discussed atomic or simple events. Simple events may trigger complex events and to capture them, event profiles can be extended to cover complex events. Event profiles provide a mechanism to weave observations of related properties to lead to identification of a complex event.

### 3.2.3 Complex Events and Profiles

In real-world applications, it may not be sufficient to determine the occurrence of an event by evaluating a single property at a specific time instance or over multiple time intervals. It may involve multiple entities and their properties evaluated at different times. To detect such *complex events* (which are generalizations of simple events), the event profiles can be defined in various ways.

The foundational concepts discussed so far provide us a way to represent various types of complexities leading to a complex event. A simple event involves one entity, one property and one observation. A complex event occurs due to multiplicity in the number of observations, properties, entities or any combination of the above.

- *Multiplicity in Observations*
  Multiple observations related to the same entity and observable property taken at different times can lead to a complex event, the profile for which may be represented as follows.

  $$P_c^o = \omega^{\kappa_1,\pi_1,\tau_1} \wedge \omega^{\kappa_1,\pi_1,\tau_2} \tag{8}$$

  For instance, the sequence of two atomic events occurring within a temporal distance is a complex event. Figure 3 depicts the scenario in which a complex event occurs based on an interpretation model that involves two observations of the same property at different times. In this scenario, the difference in temperature observations exceeds the pre-defined limit thereby triggering a temperature rise event. Note that averages over moving windows are captured in this category.

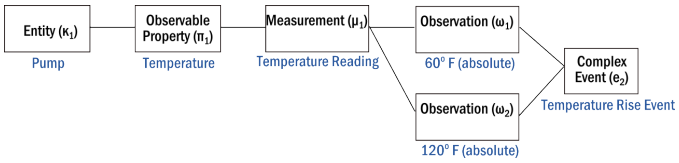


**Figure 3: Complex event caused by multiplicity in observations of same property at different times**

- *Multiplicity in Observable Properties*
  The next level of complexity might occur via different observable properties associated with the same entity, as shown by the profile below.

  $$P_c^o = \omega^{\kappa_1,\pi_1,T_1} \wedge \omega^{\kappa_1,\pi_2,T_1} \tag{9}$$

  Figure 4 represents a complex event caused by measurements in two different observable properties (temperature and pressure) of the same pump entity.

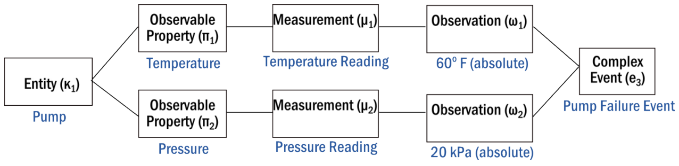


**Figure 4: Complex event involving two different observable properties for the same entity**

- *Multiplicity in Entities*
  The involvement of multiple entities, which may of the same or different type, lead to complex events. An event profile involving two different entities may simply be shown as the intersection of the two related simple events, as in 10.

  $$P_c^o = \omega^{\kappa_1,\pi_1,T_1} \wedge \omega^{\kappa_2,\pi_2,T_1} \tag{10}$$

- *Combination of any of the above*
  A combination of multiplicities in any of the above three criteria can cause a complex event. To illustrate this scenario, consider Figure 5 where the entity, observable property, and observations are all different. It indicates an interpretation where a complex event of production loss in the oilfield is triggered by multiple sub-events and observations.

Considering the above criteria, we propose a definition for complex events according to our model.

DEFINITION 2. *A complex event $e_c$ is the interpretation of an observation of interest which depends on multiple simple events. It may be represented as:*

$$e_c = X_{\omega K,\Pi,T,} \tag{11}$$

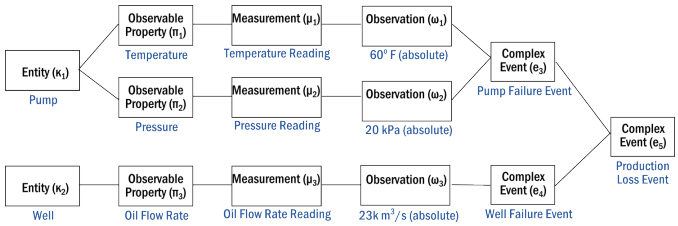


**Figure 5: Complex event involving multiple properties, entities, observations and sub-events**

**Table 1: Common Event Composition Operators**

| Operator | Syntax |
|---|---|
| Conjunction | $e_i \wedge e_j \wedge \ldots = \bigcap_{i=1}^{n} e_i$ |
| Sequence | $[e_i; e_j]_T$ |
| Disjunction | $e_i \vee e_j \vee \ldots = \bigcup_{i=1}^{n} e_i$ |
| Negation | $\bar{e_i}$ |

*where $e_c$ refers to the complex event, $T$ is the timestamp of the complex event, $K$ is the set of entities associated with the complex event, $\Pi$ is the set of observable properties associated with the complex event, and $X$ is the interpretation of the complex event. If the simple events constituting the complex event are $e_1, e_2, \ldots$ as defined in Equation 1, then $K = \bigcup_{i=1}^{n(entities)} \kappa_i$, $\Pi = \bigcup_{i=1}^{n(obsproperties)} \pi_i$ and $T$ is the timestamp of the last observation related to the complex event $T = \max\{t_i\}$ if $t_i$ are timestamps of all related observations within the complex event. $X$ is an interpretation which is based on the individual simple event interpretations through a complex event profile.*

In order to obtain complex events, we perform a composition of simple events using logical or temporal operators such as conjunction, disjunction, negation, or sequence. The algebraic semantics of complex event operators are analyzed in many CEP approaches for combining events and forming rules, and more details can be found in related event algebra works such as ETALIS [2], CERA [8], CEDL [7] and by Hinze [10]. Temporal operators are instances of a broad class of contextual operators, and other contexts, such as a spatial context, could be used instead. Table 1 lists some popular complex event operators and their notations in our model. Using a combination of these operators, it is possible to build complex rules for representing real world event processing scenarios, especially for detecting complex events. We provide a few examples of representing complex events based on the definitions and the operators introduced.

- A sequence of events within a temporal gap of $t$

$$e_{sequence} = e_i \wedge e_j \wedge (\mid \tau_i - \tau_j \mid < t)$$

- High temperature recorded for two different pumps leading to a complex event

$$e_C = e_i \wedge e_j = \chi_{\omega_i^{pump_i, temp, \tau_i,}} \wedge \chi_{\omega_j^{pump_j, temp, \tau_j}} \wedge i \neq j$$

- High temperature in pump-1 and low pressure in well-2 within 1 hour of each occurrence

$$\begin{aligned} e_C = \; & \chi_{\omega_i^{pump_1, temp, \tau_i}} \wedge \chi_{\omega_j^{well_2, pressure, \tau_j}} \\ & \wedge(\mid \tau_i - \tau_j \mid < 1 \text{ hour}) \\ & \wedge \chi_i = [\text{HighTemp}] \wedge \chi_j = [\text{LowPressure}] \end{aligned}$$

Figure 6 is an illustration of most of the foundational as well as event concepts introduced above. An important observation from the figure is that as we progress through the various elements of the model, some contextual dimensions are added to the previous concept to build a new concept. We begin with raw data values from the data stream. Measurements add the context of unit of measure, measurement type, and bounds to the data values. Observations are abstractions of measurements which deal with instantaneous measurement values, and also have a temporal context associated with them about when the value ceases to be valid. Events add further context to observations by only including those observations which have a corresponding match in an event profile and are relevant for future processing. Complex events are generalizations of simple events and include multiple events. Each of the contextual additions or enrichment can be expedited by using a semantic knowledge repository. With identification of all relevant interpretations that lead

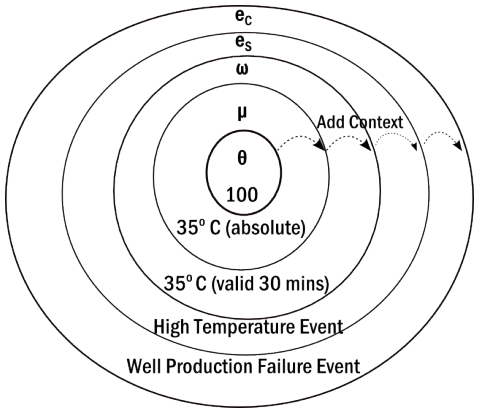


**Figure 6: From raw data values to events: Context addition and abstraction in our model**

to simple or complex events of interest, it is possible to define the *event space* as the set of all possible events that can occur in the given universe of discourse.

## 3.3 Processing Concepts

The model as described so far captures the static aspects of CEP, enabling us to model entities, properties, and their interdependence. However, these concepts are applicable at design time only. In a practical scenario, these rules should be applied over incoming streams of data. This requirement leads to various processing steps including event detection, notification, and filtering. We identify processing requirements from event profiles that leads to detection of events, states, and associated actions and processes in the application domain. Once actions are identified for the detected events, we introduce CEP concepts such as event notification and filtering.

### 3.3.1 State Model

Throughout its life-cycle, an entity goes through several states, e.g., a pump in our oilfield scenario might be in the procurement, working, failed or decommissioned states at various times. We incorporate the states of all associated entities and relationships of states to other concepts in a state model. An entity belongs to exactly one state at any point in time and has one desirable goal state (obtained from background knowledge). Entities change their states from time to time based on certain actions. A detailed description about actions can be found in Section 3.3.2. States in our model are denoted by $\psi$ for individual state instances or $\Psi$ for the set of states. The concept of states in our model is similar to "entity status" in the E* event model [9].

State-based models are frequently used in event processing. A popular choice is to use state machines or automatons, either deterministic finite automatons (DFA), or nondeterministic finite automatons (NFA), for instance, Siddhi [18] uses NFAs. However, the primary purpose of using state models (such as NFAs) in these systems is to perform complex event detection by modeling states in the system and monitoring changes in these states leading to events. We propose to use a DFA state model not for complex event detection (which we efficiently achieve through interpretations and event profiles as described in Section 3.2.2), but to manage state and action models. A state model specifies the sequence of states which should be followed so that an entity can reach its desirable goal state. It is also the basis for determining actions which are responsible for transitions between states so that the system can undertake these actions as specified by the sequence of states to reach the goal state. For the state model, we propose to use a DFA in which states of the DFA correspond to states of the entity in our case and input symbols of DFA correspond to actions. An initial start state and a final state for the entity need to be provided. A state-action transition table is also needed to complete the model which may be obtained from domain expertise or learning from historical data.

This state model approach allows identification of additional observable properties that further lead to various measurements and interpretations. It allows identification and specification of multiple properties of the entity that can be relevant during specific life-cycle phases thereby ensuring complete coverage of the modeling process. The state model also provides the additional benefit of establishing relationships with other foundational concepts. For instance, each life-cycle phase may lead to a specific set of entities and business processes. In the case of a pump in an oilfield, the state related to procurement of the pump is ascertained using specific observable properties in a procurement system or database. This also leads to identification of the procurement process itself and other entities involved, e.g. people, companies, and systems.

A state model for a pump in our oilfield scenario is given in Figure 7. Various states and actions associated with the pump are shown. The goal state (which is "working" here) is denoted by double circles. Along with the actions, the actor role who is supposed to perform the corresponding action, is also depicted. It is interesting to note that observable properties such as vibration, pressure or temperature are only relevant to the "working" state of the pump. In other phases of the pump's life-cycle, these observable properties are not applicable. For instance, while a pump is in a procurement state, its relevant observable properties may be the associated order status or shipping status. Such properties are measured, tracked and updated in different systems. Similarly, while under repairs, the pump may have a separate set of observable properties that relate to work-order information. Change in property values can lead to a change in the current interpretation, and a change in interpretation may lead to change in state.

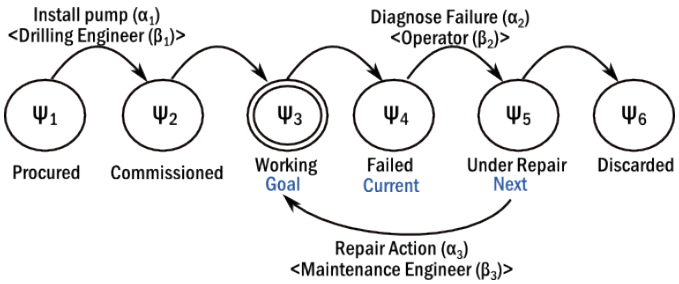


**Figure 7: State model for a pump in an oilfield**

From Figure 7 it can be noted that some of the states are desirable for an entity at a certain time while others are not. For instance, "failed" state of the pump is not desirable, hence immediately after the failed state is detected, the pump is subjected to the "maintenance" action, in order to return it back to the desirable "working" state, referred to as the "goal" state. However it may not be possible to directly achieve the goal state in a single action. Hence, it is required to determine a path to the goal state requiring a series of intermediate states. In order to accommodate these requirements, the state model should be able to capture rules for determining the goal states, next states and actions in a specific local context.

### 3.3.2 Action

A conceptual model of all possible actions requires careful identification of various types of actions in the given context. Among different kinds of actions, we have already identified actions that are relevant to changes of state for a particular entity. In Section 3.3.1, we established that actions and processes are associated with state transitions. It is possible to determine actions that lead to the desired state. Since these actions are specific to a real-world business or organizational context we identify them as *domain actions*. For instance, in the digital oilfield, actions performed by operators, maintenance engineers and other teams are domain-specific and therefore classified as domain actions.

In addition to domain actions, CEP also requires certain actions to be performed in order to detect and process events. For instance, on detection of simple events, it is required to evaluate complex event profiles. Once simple and complex events are identified, the approach should lead to identification of states and necessary processing actions (such as notification to subscribers) required in response to the detected event. These actions are identified as *CEP actions*. These types of actions are covered in most CEP approaches. A simple method to represent such actions is by using Event-Condition-Action rules.

Another type of action is required to implement the CEP actions. CEP actions provide a list of tasks to be performed

at specific time instances or intervals. However, these tasks are realized in the form of database queries, computations, etc. These sets of tasks are performed automatically in the specific system implementation, and thus identified as *system actions* or *middleware actions*.

$$action = \begin{cases} \alpha_{domain} & \text{e.g. maintenance, operations} \\ \alpha_{CEP} & \text{e.g. event detection, notification} \\ \alpha_{system} & \text{e.g. patterns life-cycle management} \end{cases} \tag{12}$$

A simple system action identified for event detection may have complex requirements. For instance, a complex event profile may require sensor readings from two sensor streams with separate update frequencies and data formats. Once these values are extracted and converted into desired form, they need to be evaluated as per the interpretation set. When an interpretation is identified, associated state and domain actions needs to be determined. An implementation of these actions therefore involves a multi-step workflow that should be realized automatically by the CEP system. In implementing these tasks, the system can employ specific enterprise integration patterns (EIPs) [12] to handle inputs/outputs such as sensor readings, temporary storage, processing and communication across software or logical entities. We have proposed semantic representations for EIPs [14]. The conceptual model should be able to make provisions for covering these issues. Upon detection of an event, the CEP system should be able to create patterns, utilize these patters in the information processing workflows and destroy patterns once processing is complete. Event escalation is a special type of CEP action, which gets triggered in exception circumstances.

### 3.3.3 Role

Based on the identified types of actions, it is possible to determine the roles of actors who are required to perform particular actions. For the identified domain actions, a person who is interested or responsible in specific actions and resulting state transitions can be identified for playing the role of an *actor*. Since the person or agent is responsible for performing specific domain actions, the role is identified as a *domain role*. In an oilfield scenario, a person responsible for maintenance of a failed pump can be considered to play a domain role of maintenance engineer.

However, just as in the case of actions, roles can be of various types and can be played by real-life or logical (software agent) entities. In case of CEP actions, the corresponding roles can be identified. For instance, the CEP engine should perform a query to determine the recipient of the notification message about the detected event. In other words, publisher and subscriber roles can be identified for a given event. It is interesting to note that an actor playing a domain role (e.g maintenance engineer) may also be assigned a *CEP role* (e.g. event subscriber). Additionally, as discussed in the context of life-cycle management of enterprise integration patterns, this conceptual modeling approach also allows life-cycle management of various roles. For instance, upon detection of an event, set of appropriate roles (like subscribers) are identified. When the event is handled appropriately, the subscriber and related roles are destroyed. The third kind of roles corresponds to *system roles* or *middleware roles* that determines various middleware actions performed by a software agent. For instance, a software agent responsible for determining subscription to events can be assigned a role of broker. Similarly a software agent responsible for generating event related information can be identified by the role of publisher. System roles can also be realized using rule-based representations.

$$role = \begin{cases} \beta_{domain} & \text{e.g. maintenance engineer, operator} \\ \beta_{CEP} & \text{e.g. publisher, subscriber} \\ \beta_{system} & \text{e.g. query manager, broker} \end{cases} \tag{13}$$

The sensor stream can be assigned a role of **event source** whereas, a particular domain role can be considered as an **event sink**. Message subscription can also be realized in various ways to subscribe software agents and human beings appropriately. Subscriptions can be implemented by selecting proper enterprise integration patterns. Equation 14 defines rules to determine appropriate patterns while determining subscription for human or software agents.

IF Software agent THEN Constant Polling
IF Human agent THEN Request-Reply/Publish-Subscribe (14)

### 3.3.4 Notification

When an event ($e$) is detected, and corrective actions ($\alpha$) and roles ($\beta$) are identified, the next processing step is *event notification*. Here, the CEP system should be able to determine the content of the notification message $N$ shown in Equation 15. Event notification is typically limited to core event detection information but we introduce additional background information to enrich the notification message. For instance, each corrective action ($\alpha$) can be associated with an expected time for completion $\tau_{comp}$. This not only provides guidance to the actor but also enables a mechanism for validation of the notification. The CEP system can evaluate the state of an entity associated with the event to determine changes. If the action is not taken and entity remains in the same state, the CEP system may trigger additional actions to escalate the event. In addition to expected time of completion, the CEP system might include best practices related to the required action and additional information that is available in the background knowledge base ($KB$).

$$N = \langle \beta, \alpha, \tau_{comp}, \text{event-context} \rangle \tag{15}$$

Using this approach it is possible to determine how long a subscription is valid. Subscriptions identified for an event can be maintained until the goal state is achieved. If the requisite action is not performed on the first notification, an additional escalation message is generated. Here, the role $\beta$ and action $\alpha$ can change according to the escalation process adopted in the specific enterprise. Similarly, when a message is escalated, the granularity of its content can also change based on the rules. As subscriptions and notifications are managed in a rule-driven process, it leads to possibility of redundancies and duplication. This leads us to identify provisions for filtering mechanisms.

### 3.3.5 Filtering

Filtering mechanisms play a key role in avoiding duplicate or unwanted occurrences of instances to report unique happenings from the deluge of observations. An event profile evaluated at certain time intervals may detect an event that triggers actions, roles and notifications. However, at the next time interval, it might detect the same event again

and generate duplicate notifications. In case of the oilfield scenario, suppose an event profile to determine pump failure is evaluated every minute. If a failure event is detected at time $\tau_i$, the system would detect the same event again at $\tau_{i+1min}$ (unless the failure is fixed within such a short time) resulting in redundancy. To avoid this situation, filtering can be employed. Existing patterns for event filtering, as mentioned by Paschke et al. [13] can be used here.

## 3.4 Event Processing Workflow

The complete event processing workflow enabled by the proposed model is depicted in Figure 8. The top half of the figure delineates various elements of the knowledge base (KB) to be looked up by the components involved in event processing at runtime, which are shown in the bottom half. The process begins with reading values from data streams. Simple as well as complex events can be detected by using the respective event profiles and reporting if any interpretations within the event profile are evaluated to be true (event-to-profile match is found, similar to the "event matching" operator in [8]). An event ($e$) is detected in such a case. Once an event is detected, the current state ($\psi_{current}$) and goal state ($\psi_{goal}$) of the entity involved are determined. If they are the same, then the entity is in the desired state, and no further processing needs to be done. Thus, the event and its related information can be discarded (after logging, if necessary). If the entity is not in the goal state, a sequence of states which needs to be traversed to reach the goal state is determined by referring to the state transition model. The first state from this sequence is the next state, and an action ($\alpha$) leading to the next state is also looked up. After obtaining the action, the action-role mapping is looked up in the KB to determine appropriate roles ($\beta$) of actors who can perform the specific action. From the role, particular actors, e.g. employees who need to perform the action, are found. Finally, a notification message is sent to the actors (as well as any other event subscribers), the content of the message being enriched with relevant context. However, the event processing workflow does not end here, since it is unknown if the action suggested was actually performed. In such cases, escalation may need to be performed to ensure that appropriate action is taken and the entity reaches its desired goal state.

## 3.5 Event Escalation Scenario

Escalation refers to the case when an event is detected, a notification is sent to an actor to take a certain action but no reply from the actor is received (probable reasons for which may be the actor being inactive, the actor being unable to perform the action or the action being performed but the reply message getting lost). Conventional event models would stop processing the event here, or retry transmission of the action message, and wait till a response is received or someone finds out the root cause for the problem. It would be better if a proactive CEP system could monitor the repair action suggested to the actor, and upon exceeding a certain expected time to reply, would automatically *escalate* the event and perform a set of predefined actions (which could include sending a message to the supervisor of the actor or another actor/team to carry out the repair action, or aborting the event altogether). Such predefined actions can be determined from background knowledge and information about the organization and use case domain.

A step-by-step algorithm for an event processing scenario which includes event escalation is depicted in Algorithm 1. The process involves several lookup operations to obtain information from the knowledge base (KB). The algorithm shown assumes a simple event. The detected event has an associated entity, timestamp and property, as defined in Definition 1. The process can be extended to a complex event by replacing the individual instances of entities and properties with their corresponding sets.

Within the escalation loop, the algorithm begins by obtaining an estimated time for completion of the action. A message is sent to the actors with appropriate roles with details about the entity, event and this time value. The system then waits for this amount of time and then looks up associated escalation roles who can perform or manage the required action. This new set of roles is added to the role set which serves as the subscriber list for the action notification message. By including the set of roles instead of a single role, it is ensured that any actor can perform the required action and that all of them are kept coordinated about the results. If the state of the entity has not reached the desirable next state, this process is repeated. Otherwise, the escalation process is canceled and the program returns execution to the outer loop to progress further in the state model towards the goal state. If the nature of the event is such that the goal state may never be reached, appropriate error messages can be passed to the user and escalation can be terminated.

**Algorithm 1** Event Escalation Algorithm

```
e ← χ_{ω^{τ,κ,π}}
Get ψ_current, ψ_goal
while ψ_current ≠ ψ_goal do
    ψ_next ← LookupNextState(ψ_current, κ)
    α ← LookupAction(ψ_current, ψ_next)
    β ← LookupRole(α)
    B ← {β}
    repeat
        Get estimated t_α
        SendMsg(α, β, t_α, e)
        Sleep(t_α)
        B ← B ∪ LookupEscalationRole(B, e)
    until ψ_current ≠ ψ_next
    ψ_current ← ψ_next
end while
```

# 4. CASE STUDIES

## 4.1 The PoEM Ontology

We developed an ontology for the PoEM model closely following the described concepts. Each concept translates to an OWL class in the ontology and inter-relationships between concepts are represented by properties. Figure 9 shows an overview of the ontology schema[5]. Having an ontology for PoEM enables us to connect to open linked data. For instance, time-related concepts can be borrowed from the W3C Time ontology[6], and geospatial concepts can be reused from the Geonames ontology[7]. It also enables us to

[5]The ontology was developed using Topbraid Composer™
[6]`http://www.w3.org/TR/owl-time/`
[7]`http://www.geonames.org/ontology/`

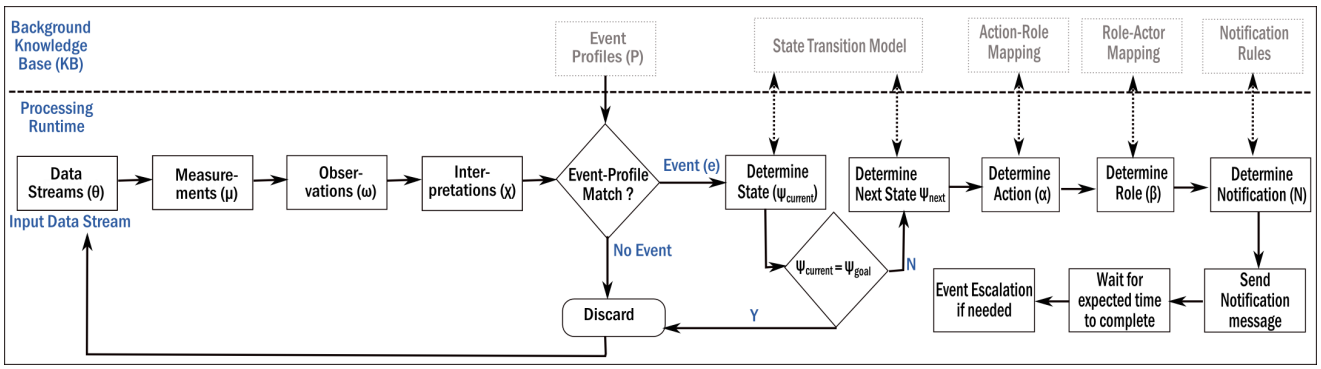


**Figure 8: PoEM Event Processing Workflow**

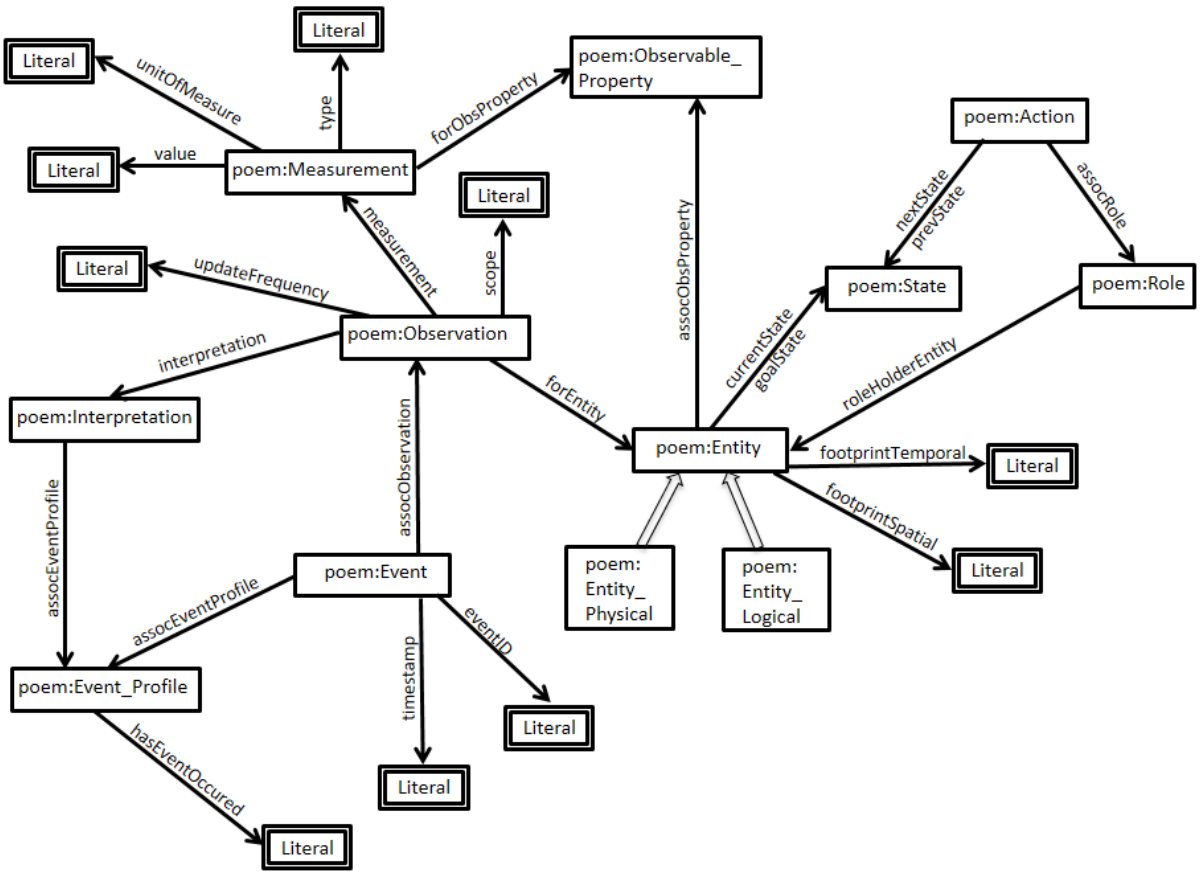


**Figure 9: PoEM Ontology Classes and Properties. Classes are shown as rectangles and literals (resulting from datatype properties) are shown in double rectangles.**

compare against existing models, such as the Simple Event Model (SEM) [20] and provide cross-links to other models.

## 4.2 Oil and Gas Industry Use Case

Using the PoEM ontology, we model the pump failure scenario in an oilfield described in Section 1.1. As shown in Figure 10, consider a pump (entity in PoEM) which has sensors to constantly monitor the pressure by the load. Excessively high or low pressure values can lead to anomaly events. Consider an event detection rule to detect rapid rates of change in the pressure value of the pump. Using the concepts of measurements, observations and interpretations described earlier, such a high pressure event can be detected, serving as an indicator of pump failure. The state-based goal planning method is triggered next, leading to identification of current and goal states, and the sequence of actions required to move the pump to the goal state. The associated roles with the actions are queried and relevant personnel such as maintenance engineers are notified. The event context should be stored as the same notification can be sent to the production team as well so as to avoid application or system silos. The proposed event escalation mechanism ensures reduced response times and ultimately, higher oil production. Even though a complex event resulting from multiplicity in observations is shown here, more complex scenarios are possible as described earlier.

## 4.3 Piracy Events

To further illustrate the applicability of the PoEM model, we use the example of modeling maritime piracy events, introduced by Van Hage et al. [20] to illustrate the Simple Event Model. We represent the same domain using the PoEM model by a simple identification and mapping of piracy event related concepts to PoEM ontology classes. This mapping is shown in Figure 11. The mapping process is similar to that used for the O&G use case. Note that PoEM enables adding more context than the originally intended purpose of modeling just a piracy event detection scenario. Using PoEM, we can also model event notifications, actions, roles, and escalation to complete the event processing cycle. The maritime piracy use case is described in [20].

# 5. CONCLUSIONS AND FUTURE WORK

We observed that existing generic event modeling approaches do not provide a means of representing an end-to-end event processing workflow. We designed a conceptual model, PoEM, to be comprehensive while not being restricted to one application domain. We introduced the foundational, event and processing concepts for building the PoEM model. PoEM includes state-based goal planning and event escalation. We illustrated the applicability of PoEM to industrial processes, specifically, managing failure events in an oilfield and modeling maritime piracy events.

The event model presented here is limited to identification of key concepts and relationships between them. We currently assume that all data is available at a central location holding decision-making capability, and will extend PoEM to account for distributed data sources. We will focus on the use of machine learning approaches to develop the background knowledge required to implement the PoEM model. We will explore semantic web techniques to include rule-based reasoning, and event-based middleware and other implementation aspects to support the proposed model.

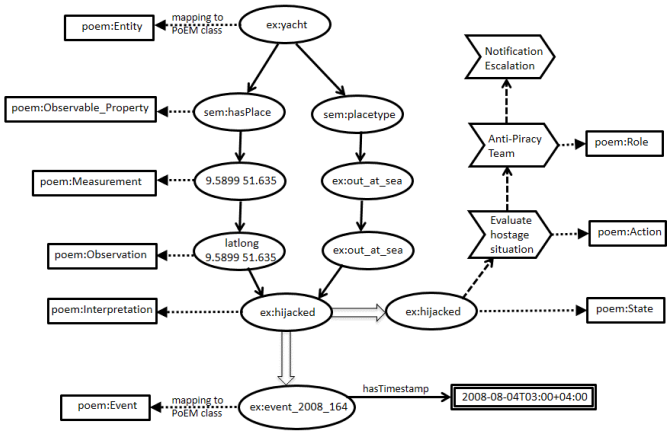


**Figure 11: Maritime piracy events mapped to PoEM. `sem:` denotes the simple event model ontology [20] and `ex:` is a maritime domain ontology.**

# 6. ACKNOWLEDGMENTS

This work is supported by Chevron Corp. under the joint project, Center for Interactive Smart Oilfield Technologies (CiSoft), at the University of Southern California.

# 7. REFERENCES

[1] A. Abou-Sayed. Data mining applications in the oil and gas industry. *Journal of Petroleum Technology*, 64(10):88–95, 2012.

[2] D. Anicic, P. Fodor, S. Rudolph, R. Stühmer, N. Stojanovic, and R. Studer. Etalis: Rule-based reasoning in event processing. *Reasoning in EventBased Distributed Systems*, 347:99–124, 2011.

[3] R. Blanco, J. Wang, and P. Alencar. A metamodel for distributed event based systems. In *Proceedings of the second international conference on Distributed event-based systems*, pages 221–232. ACM, 2008.

[4] K. Chandy, M. Charpentier, and A. Capponi. Towards a theory of events. In *Proceedings of the 2007 inaugural international conference on Distributed event-based systems*, pages 180–187. ACM, 2007.

[5] K. M. Chandy, O. Etzion, and R. von Ammon. The event processing manifesto. In *2010 Dagstuhl Seminar on Event Processing*, 2010.

[6] G. Cugola and A. Margara. Processing flows of information: From data stream to complex event processing. *ACM Comput. Surv.*, 44(3):15:1–15:62, June 2012.

[7] I. David. A model-driven approach for processing complex events. *arXiv preprint arXiv:1204.2203*, 2012.

[8] M. Eckert, F. Bry, S. Brodt, O. Poppe, and S. Hausmann. Two semantics for CEP, no double talk: Complex event relational algebra (CERA) and its application to XChangeEQ. *Reasoning in Event-Based Distributed Systems*, pages 71–97, 2011.

[9] A. Gupta and R. Jain. Managing event information: Modeling, retrieval, and applications. *Synthesis Lectures on Data Management*, 3(4):1–141, 2011.

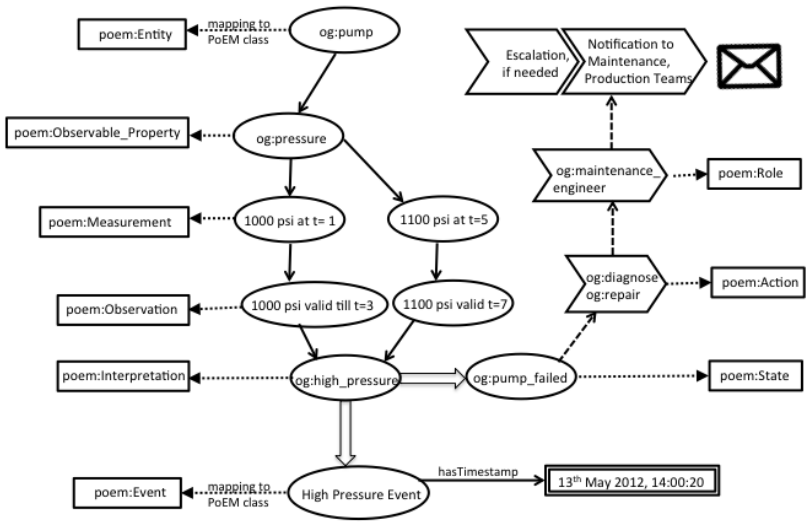


**Figure 10: Pump failure event in PoEM.** `og:` **denotes an oil and gas domain ontology.**


[10] A. Hinze. *A-MEDIAS: Concept and Design of an Adaptive Integrating Event Notification Service*. PhD thesis, Freie Universitaet Berlin, 2003.

[11] A. Hinze, K. Sachs, and A. Buchmann. Event-based applications and enabling technologies. In *Proceedings of the Third ACM International Conference on Distributed Event-Based Systems*, DEBS '09, pages 1–15, New York, NY, USA, 2009. ACM.

[12] G. Hohpe and B. Woolf. *Enterprise integration patterns: Designing, building, and deploying messaging solutions*. Addison-Wesley Professional, 2004.

[13] A. Paschke, P. Vincent, A. Alves, and C. Moxey. Tutorial on advanced design patterns in event processing. In *Proceedings of the 6th ACM International Conference on Distributed Event-Based Systems*, DEBS '12, pages 324–334, New York, NY, USA, 2012. ACM.

[14] O. P. Patri, A. V. Panangadan, V. S. Sorathia, and V. K. Prasanna. Semantic management of enterprise integration patterns: A use case in smart grids. In *10th International Workshop on Information Integration on the Web (IIWeb), IEEE 30th International Conference on Data Engineering (ICDE)*, 2014.

[15] O. P. Patri, V. S. Sorathia, and V. Prasanna. Event-driven information integration for the digital oilfield. In *SPE Annual Technical Conference and Exhibition*. Society of Petroleum Engineers, 2012.

[16] R. Shaw, R. Troncy, and L. Hardman. Lode: Linking open descriptions of events. *The Semantic Web*, pages 153–167, 2009.

[17] V. Sorathia. *Dynamic Information Management Methodology with Situation Awareness Capability*. PhD thesis, Dhirubhai Ambani Institute of Information and Communication Technology, 2008.

[18] S. Suhothayan, K. Gajasinghe, I. Loku Narangoda, S. Chaturanga, S. Perera, and V. Nanayakkara. Siddhi: a second look at complex event processing architectures. In *Proceedings of the 2011 ACM workshop on Gateway computing environments*, pages 43–50. ACM, 2011.

[19] K. Teymourian, G. Coskun, and A. Paschke. Modular upper-level ontologies for semantic complex event processing. In *Proceedings of the Fourth International Workshop on Modular Ontologies*, pages 81–93. IOS Press, 2010.

[20] W. Van Hage, V. Malaisé, R. Segers, L. Hollink, and G. Schreiber. Design and use of the simple event model (SEM). *Web Semantics: Science, Services and Agents on the World Wide Web*, 2011.

[21] A. Voisard and H. Ziekow. Architect: A layered framework for classifying technologies of event-based systems. *Information Systems*, 36(6):937–957, 2011.

[22] S. Wasserkrug, A. Gal, O. Etzion, and Y. Turchin. Efficient processing of uncertain events in rule-based systems. *Knowledge and Data Engineering, IEEE Transactions on*, 24(1):45–58, 2012.

[23] M. Weidlich, J. Mendling, and A. Gal. Net-based analysis of event processing networks – the fast flower delivery case. In *Application and Theory of Petri Nets and Concurrency*, pages 270–290. Springer, 2013.

[24] Q. Zhou, Y. Simmhan, and V. Prasanna. Towards hybrid online on-demand querying of realtime data with stateful complex event processing. In *Big Data, 2013 IEEE International Conference on*, pages 199–205. IEEE, 2013.

[25] D. Zimmer and R. Unland. On the semantics of complex events in active database management systems. In *Data Engineering, 1999. Proceedings., 15th International Conference on*, pages 392–399. IEEE, 1999.