Information of Paper: A Representation of Event-Driven Systems based on Pre-conceptual Schemas

[1], [2], [3], [4], [5], [6], [7] [8], [9], [10], [11], [12]. [13], [14] [15], [16], [17], [18] [19], [20], [21], [4], [22], [19], [23], [24], [25], [19], [20], [21], [22], [23], [24]

1 BACKGROUND

1.1 Event-Driven Systems

Event-driven systems or EBS are distributed computing systems that encode complex business logic driven by events [26]. In EBS, the communication and processing of events define the structure for proving the solution. One of the main benefits of the EBS is interoperability among systems, allowing to have information updated and notifications about of happens in the system [4].

Events Events are both a fact and a notification message [27]. They represent something that happens in a specific space or time in a system or domain, *e.g.* TemperatureAppears, DataArrived, TimePasses, OrderCreated, CustomerDetailsUpdated [1], [27], [28], [29]. Either a single event or a sequence of events can require a reaction or be ignored in the processes of a system. Triggered events are responsible for the system behavior and functionality by changing the state of automated and human processes [25], [30]. Events also carry instant information [31], which is relevant for specifying an information system [32]. We can classify events into two main categories: simple events and complex events [4].

Simple events (Also called primitive or atomic events) happen at one point in time [4] [2], *e.g.*, the event *temperature value changes* is represented as single data containing the measured temperature and time of measurement, *i.e.*, *event 1 = [07:00,* $10°C$*]*, *event 2 = [07:30,* $12°C$*]*, *event 3 = [08:00,* $15°C$*]* and *event 4 = [09:00,* $15°C$*]*.

Complex events Provide information over a set of events [4], and are composed of a combination of simple events coming from either a single source or many sources [2]. Complex events allow for executing business processes and generating a huge volume of data [33]. In fact, such events can be derived as outputs from a matching process [4], *e.g., event 1 = [(area, "area1"), (temperature, 25), (wind, 15)]*, in this case, the complex events combine different data values and satisfy the conditions of the filters [17].

Complex Event Processing. Complex event processing (CEP) is a set of techniques and tools for detecting complex events in real time and reacting to them. CEP offers an abstraction layer, which is used for hiding the complexity of event detection. The business-level applications are notified about the occurrence of the event in order to be focused on realizing appropriate actions whenever a specific event occurs [28].

EBS presents an event-driven architecture using the Publisher/Subscriber (PS) infrastructure [26] (see Figure 1), which can also be used in services-based architecture [27]. PS is a communication paradigm where publishers and subscribers communicate with each other by using message events [32]. Thus, both the publisher and subscriber are responsible for processing events [34]. Such architecture is based on the following elements: A *publisher* is an entity or class of an event processing system, which introduces events into the systems. A *subscriber* is an entity or class of an event processing system that receives events from the systems. Both publishers and subscribers are classified into three categories: hardware (sensors, cameras, detectors, physical actuators, industrial control, lighting system, home automation), software (simulated software, adapters, event logs, business applications, business processes, state machines), and human interaction (application programs, alarm systems, e-mails, SMS, phone and computer interfaces, new feeds). An *event processing* is a software module for processing events. An *input event* (also called *incoming event*) is introduced in the event processing system by a publisher. A *output event* (also called *derived event*) is generated in the event processing system by a publisher. Derived events can be processed again, so they can also be viewed as input events [1].

Matching (also referred to as *forwarding*) is the core functionality realized by a PS infrastructure within CEP of EBS [17]. Event matching is used for performing three logical steps (see Figure 1) within a specified context: *(i) filtering*, where relevant events from the input events are selected for processing; *(ii) matching*, where all events are analyzed by using an event processing pattern or some other kind of matching criteria; and *(iii) deriving*, where analyzed events are used for deriving the output events [16].

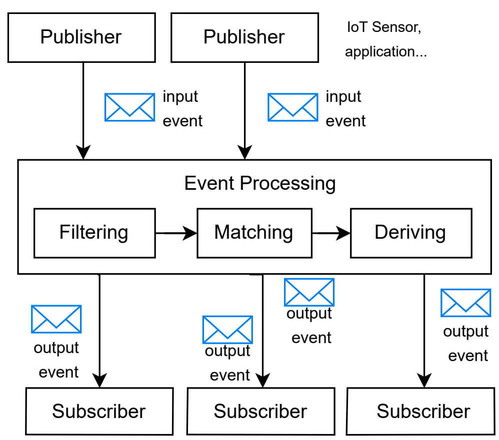

Fig. 1. PS infrastructure within CEP. The Authors based on [16]

Subscriptions, filters, and *predicates* are elements of the data model commonly used among event-based systems for event matching. Proposed algorithms for efficient event matching are conceived for running on sequential hardware by using such elements [17]. A *Subscriber* creates a subscription to indicate interest events produced by *publishers*. Each subscription is composed of a disjunction of filters, so a subscription will be matched if any single filter it contains is matched. *Filters* are a conjunction of Boolean predicates, so a filter *matches* an event only if all *predicates* are satisfied [35]. Finally, Boolean predicates are conditions or constraints on a single attribute [32].

An *event* (both *input* and *output* according to the CEP architecture elements) is a set of attributes and values. For example, *event 1* is an event from an environmental monitoring system, which could be published for notifying about the current temperature and wind speed in the monitored area, *event 1 = [(area, "area1"), (temperature, 25), (wind, 15)]* [17]. We can define *P1 = (temperature > 30), P2 = (wind > 20), P3 = (area = "area"), P4 = (area = "area2")* as potential predicates, and create filters *F1 = [(area = "area1")* $\wedge$ *(temperature > 30)* and *F2 = [(area = "area2")* $\wedge$ *(wind > 20)]*. These filters can also be represented as *F1 = (P3* $\wedge$ *P1)* and *F2 = (P4* $\wedge$ *P2)*. We can then create a subscription *S1 = [(area = "area1")* $\wedge$ *(temperature > 30)]* $\vee$ *[(area = "area2")* $\wedge$ *(wind > 20)]* as a composition of these two filters, which can also be represented as *S1 = (F1* $\vee$ *F2)*.

1.2 Domain

A domain defines the specific real-world problem that trying to solve using a software system. Effective communication between developers and domain experts is essential for the solution. A common language, which is represented in the domain and its boundary is very important, to avoid future problems and develop successful software with knowledge of the domain coherent with of software system. The abstraction of the domain is performed using a model to understand this knowledge. In a software system, such a domain is represented by elements such as processes, events, entities (concepts or classes and their attributes), requirements, business logic (constraints), etc. When the software system is closer to the domain, then such a system is closer to the client.

In architecture models such as clean architecture and domain-driven design or DDD, the domain is the heart of a software system. Therefore, the domain becomes a layer in the architecture, which is responsible for representing business information [36] [37]. For example, in the clinical surgery domain required entities *i.e.*, medical Surgeon, anesthesiologist, surgical assistant, surgery, instrumental, and patient; requirements *i.e.*, surgery stars, insert patient, read vital signs (sensor), etc.; and business rule *i.e.*, surgery discount, etc.

1.3 Pre-conceptual Schemas

Pre-conceptual schemas (PCS) are computational and graphical models used for representing the domain knowledge of a software system. PCS have an intuitive nature for recognizing concepts and relationships in any domain [11]. Such schemas include a behavioral and structural view, which allows a complete model to be drawn based on software engineering processes and computational linguistics [11].

PCS notation. PCS notation includes components with linguistic, mathematical, and graphical structures for modeling processes and events in a domain [11], [12], [33]. Such components are divided into four groups (see Figure 2):

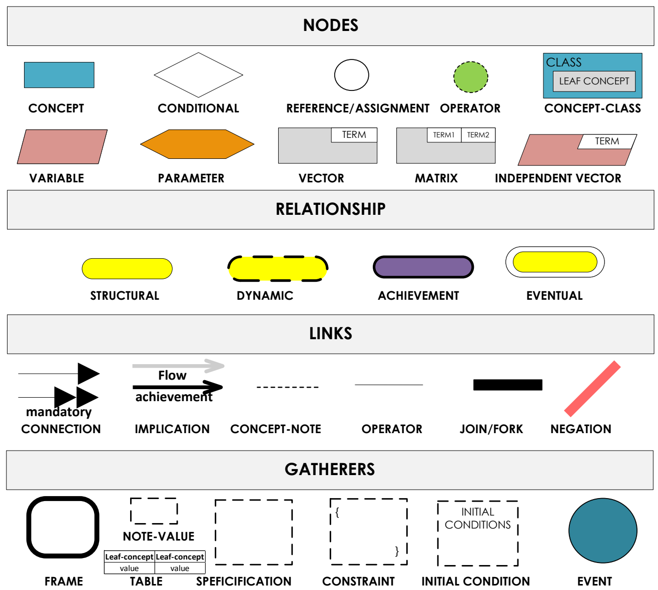

Fig. 2. PCS notation, which includes different elements for representing a domain. The Authors based on [11], [12]

Nodes

- *Concept* is used for representing a class concept (*e.g., cow* and *milk* in Figure 3), and a leaf concept/attribute (*e.g., amount* in Figure 3).
- *Conditional* is used for defining instructions, *e.g.,* if *milk.amount* $<=$ *30 liters* in Figure 3.
- *Reference* is used for relating a distant node by using a number.
- *Operator* is used for representing mathematical operations. Type operators are: *logical operators* (AND, OR); *arithmetic operators* $(+, -, *, /)$; and *relational operators* $(<, >, =, !=, <=, >=)$, *array operators (push* for storing data to an array and *pop* for deleting them), and *complex operators* (*log* for logarithm function, *sin* for sine, trigonometric function, *rand* for random function, etc. and functions added by an analyst), *e.g.,* relational operator $(<=)$ in Figure 3.
- *Concept-class* is used for representing a class with its leaf concept *e.g., milk.amount* in Figure 3.

- *Variable* is used for representing variable values, *e.g., thermometer state* variable.
- *Parameter* is used for representing constant values, *e.g., PI* constant.
- *Vector*, *matrix*, and *independent vector* are arrays or data structures. Their index or position is called "term" [12, 33].

Relationships

- *Structural relationship* is used for relating a class concept and its leaf concepts by using the verb "has," (*e.g.*, milk has amount in Figure 3), and defining an inheritance by using the verb "is," *e.g.*, user is analyst.
- *Dynamic relationship* is used for representing a process or a service, *e.g., milker collects milk* in Figure 3.
- *Achievement relationship* is used for representing objectives, *e.g.*, improving security.
- *Eventual relationship* is used for representing events *e.g., customer arrives* in Figure 3.

Links

- *Connection* is used for relating nodes and relationships.
- *Implication* is used for relating dynamic relationships, conditionals, and events, indicating a flow in the system.
- *Concept-note* is used for relating values, specifications, and constraints.
- *Operator* is used for relating operators, concepts, and values.
- *Joint/fork* is used for relating implication links.

Gatherers

- *Frame* is commonly associated with reports.
- *Note-Value* is an assignation value of nodes.
- *Table* is a gatherer of possible data.
- *Specification* is used for including values and operations in the internal logic of processes and events.
- *Constraint* is also a specification including values and operations with conditions.
- *Initial conditions* is a gatherer used for starting variables and parameters of the system.
- *Event* is used for triggering dynamic relationships and other events.

A PCS in an agricultural domain is presented as a simple example in Figure 3. *Cow has id*, *cow has name*, and *milk has amount* are structural relationships, and *cow* and *milk* are classes. *Id, name,* and *amount* are leaf concepts (attributes). *Cow produces milk, milker collects milk,* and *seller sells milk* are dynamic relationships (processes). *Milk.amount <= 30 liters* is a conditional event, *milk.amount* is a concept-class (including its attribute), <= is a relational operator, *30 liters* is a value, and *customer arrives* is an event. When both events happen the process *seller sells milk* is triggered.

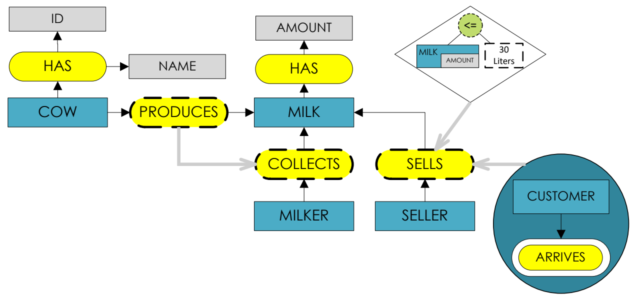

Fig. 3. Pre-conceptual schema for showing concepts, structural and dynamic relationships, a conditional, and an event in an agricultural domain. The Authors based on [38]

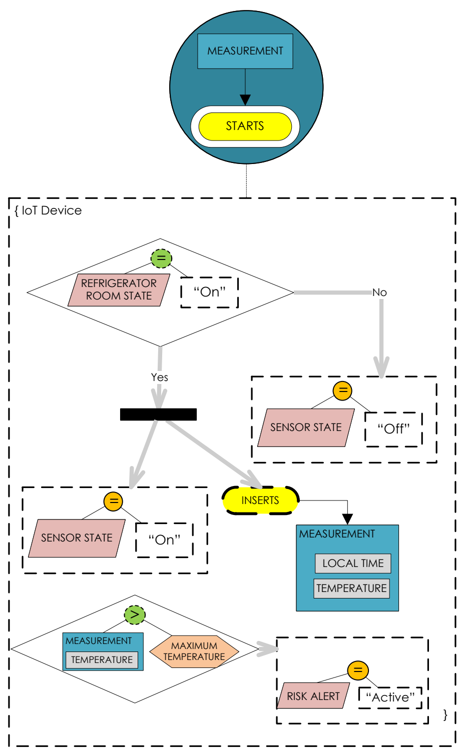

Fig. 4. Event notation in PCS. The Authors based on [12]

Event representation in PCS PCS notation allows for representing events by using *specifications* and *constraints* (see Figure 2). Both specifications and constraints include conditions, parameters, variables, functions, classes, and leaf concepts involved in events [12], [39]. We present an example in Figure 4 for explaining the event representation based on PCS [33]. An event *measurement starts* is related to its logic by a *constraint* in a simulation. When conditions are met, then the *refrigerator room state* variable changes to *"on"*, then sensor state also changes *"on"* and the *local time* and *temperature* of *measurement* is inserted. If *measurement.temperature* is greater than the *maximum temperature* then the *risk alert* is activated.

References

1. O. Etzion, P. Niblett, and D. Luckham, *Event Processing in Action.* Stanford: Manning Publications Co, 2011.
2. P. Soffer, A. Hinze, A. Koschmider, H. Ziekow, C. Di Ciccio, B. Koldehofe, O. Kopp, A. Jacobsen, J. Sürmeli, and W. Song, "From event streams to process models and back: Challenges and opportunities," *Information Systems*, vol. 81, pp. 181–200, 2019.
3. E. Brazález, H. Macià, G. Díaz, V. Valero, and J. Boubeta-Puig, "Pits: an intelligent transportation system in pandemic times," *Engineering applications of artificial intelligence*, vol. 114, p. 105154, 2022.
4. O. P. Patri, V. S. Sorathia, A. V. Panangadan, and V. Prasanna, ""the process-oriented event model (poem): A conceptual model for industrial events"," in *Proceedings of the 8th ACM International Conference on Distributed Event-Based Systems*, pp. 154–165, ACM, 2014.
5. G. Ortiz, I. Castillo, A. Garcia-de Prado, and J. Boubeta-Puig, "Evaluating a flow-based programming approach as an alternative for developing cep applications in iot," *IEEE Internet of Things Journal*, vol. 9, no. 13, pp. 11489–11499, 2021.
6. J. Roldán-Gómez, J. M. del Rincon, J. Boubeta-Puig, and J. L. Martínez, "An automatic unsupervised complex event processing rules generation architecture for real-time iot attacks detection," *Wireless Networks*, pp. 1–18, 2023.
7. J. Rosa Bilbao, J. Boubeta Puig, *et al.*, "Model-driven engineering for complex event processing: A survey," 2022.
8. OMG, *Business Process Model And Notation 2.0.* OMG, Object Management Group, 2014. `https://www.omg.org/spec/BPMN/About-BPMN/`.
9. OMG, Object Management Group, *Superstructure 2.5*, 2015. http://www.omg.org/spec/UML/2.5/.
10. S. Xue, B. Wu, and J. Chen, "Lightepc: A formal approach for modeling personalized lightweight event-driven business process," in *IEEE International Conference on Services Computing*, IEEE, 2013. Santa Clara, US, 1–8.
11. C. M. Zapata, *The UNC-Method Revisited: Elements of the New Approach.* Saarbrücken: Lambert Academic Publishing, 2012.
12. P. A. Noreña, *An Extension to Pre-conceptual Schemas for refining Event Representation and Mathematical Notation.* PhD thesis, Universidad Nacional de Colombia, Medellín Campus, Colombia, 2020.
13. G. Decker, A. Grosskopf, and A. Barros, "A graphical notation for modeling complex events in business processes," in *11th IEEE international enterprise distributed object computing conference (EDOC 2007)*, pp. 27–27, IEEE, 2007.

14. S. Kunz, T. Fickinger, J. Prescher, and K. Spengler, "Managing complex event processes with business process modeling notation," in *International Workshop on Business Process Modeling Notation*, pp. 78–90, Springer, 2010.
15. A. Barros, G. Decker, and A. Grosskopf, "Complex events in business processes," in *International Conference on Business Information Systems*, pp. 29–40, Springer, 2007.
16. I. Kolchinsky, I.and Sharfman and A. Schuster, "Lazy evaluation methods for detecting complex events," in *Proceedings of the 9th ACM International Conference on Distributed Event-Based Systems*, pp. 34–45, ACM, 2015.
17. C. Balkesen, N. Dindar, M. Wetter, and N. Tatbul, "Rip: Run-based intra-query parallelism for scalable complex event processing," in *Proceedings of the 7th ACM international conference on Distributed event-based systems*, pp. 3–14, ACM, 2013.
18. C. Reinartz, A. Metzger, and K. Pohl, "Model-based verification of event-driven business processes," in *Proceedings of the 9th ACM International Conference on Distributed Event-Based Systems*, pp. 1–9, ACM, 2015.
19. J. Ollesch, M. Hesenius, and V. Gruhn, "Engineering events in cps-experiences and lessons learned," in *2017 IEEE/ACM 3rd international workshop on software engineering for smart cyber-physical systems (SEsCPS)*, pp. 3–9, IEEE, 2017.
20. A. Bauer and C. Wolff, "An event processing approach to text stream analysis: Basic principles of event based information filtering," in *Proceedings of the 8th acm international conference on distributed event-based systems*, pp. 35–46, ACM, 2014.
21. L. Renners, R. Bruns, and J. Dunkel, "Situation-aware energy control by combining simple sensors and complex event processing," in *Workshop on AI Problems and Approaches for Intelligent Environments in conjunction with ECAI 2012*, pp. 33–38, 2012.
22. J. Bruns, R.and Dunkel, S. Lier, and H. Masbruch, "Ds-epl: Domain-specific event processing language," in *Proceedings of the 8th ACM International Conference on Distributed Event-Based Systems*, pp. 83–94, ACM, 2014.
23. O. Etzion, I. Fournier, F.and Skarbovsky, and B. von Halle, "A model driven approach for event processing applications," in *Proceedings of the 10th ACM International Conference on Distributed and Event-based Systems*, pp. 81–92, ACM, 2016.
24. J. Boubeta-Puig, H. Díaz, G.and Macià, V. Valero, and G. Ortiz, "Medit4cep-cpn: An approach for complex event processing modeling by prioritized colored petri nets," *Information Systems*, vol. 81, pp. 267–289, 2019.
25. P. A. Noreña and C. M. Zapata, "A game for learning event-driven architecture: Pre-conceptual-schema-based pedagogical strategy," *Developments in Business Simulation and Experiential Learning*, vol. 45, pp. 24–37, 2018.
26. W. Hummer, C. Inzinger, P. Leitner, B. Satzger, and S. Dustdar, "Deriving a unified fault taxonomy for event-based systems," in *6th ACM International Conference on Distributed Event-based Systems*, pp. 167–178, 2012.
27. B. Stopford, *Designing Event-Driven Systems.* O'Reilly Media, Incorporated, 2018.
28. D. Luckham, *The Power of Events. An Introduction to Complex Event Processing in Distributed Enterprise Systems.* Boston: Addison-Wesley, 2002.
29. A. A. Jalbani, *Quality Assessment and Quality Improvement for UML Models.* PhD thesis, Niedersächsische Staats-und Universitätsbibliothek Göttingen, 2011.
30. P. A. Noreña and C. M. Zapata, "Una representación basada en esquemas preconceptuales de eventos determinísticos y aleatorios tipo señal desde dominios de software científico," *Research in Computing Science*, vol. 147, no. 2, pp. 207–220, 2018.

31. D. Luckham, *Event processing for business: organizing the real-time enterprise.* John Wiley & Sons, 2011.
32. J. Munster and H.-A. Jacobsen, "Secret sharing in pub/sub using trusted execution environments," in *Proceedings of the 12th ACM International Conference on Distributed and Event-based Systems*, pp. 28–39, ACM, 2018.
33. P. A. Noreña and C. M. Zapata, "Business simulation by using events from pre-conceptual schemas," in *Developments in Business Simulation and Experiential Learning: Proceedings of the Annual ABSEL conference*, vol. 46, pp. 258–263, 2019.
34. P. Salehi, K. Zhang, and H.-A. Jacobsen, "Popsub: Improving resource utilization in distributed content-based publish/subscribe systems," in *Proceedings of the 11th ACM International Conference on Distributed and Event-based Systems*, pp. 88–99, ACM, 2017.
35. A. Margara and G. Cugola, "High-performance publish-subscribe matching using parallel hardware," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 1, pp. 126–135, 2014.
36. R. C. Martin, *Clean Architecture: A Craftsman's Guide to Software Structure and Design.* Boston: Addison-Wesley, 2018.
37. V. Vernon, *Implementing Domain-Driven Design.* Boston: Addison-Wesley, 2013.
38. J. S. Zapata-Tamayo and C. M. Zapata-Jaramillo, "Pre-conceptual schemas: Ten years of lessons learned about software engineering teaching," *Developments in Business Simulation and Experiential Learning*, vol. 45, pp. 250–257, 2018.
39. P. A. Noreña, C. M. Zapata, and A. E. Villamizar, "Representing chemical events by using mathematical notation from pre-conceptual schemas," *IEEE Latin America Transactions*, vol. 17, no. 01, pp. 46–53, 2019.