View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by Open Access LMU

Complex Event Processing (CEP)

Michael Eckert and François Bry
Institut für Informatik, Ludwig-Maximilians-Universität München
michael.eckert@pms.ifi.lmu.de, http://www.pms.ifi.lmu.de

This is an English translation of the article "Aktuelles Schlagwort: Complex Event Processing (CEP)" (to appear in German language in Informatik-Spektrum, Springer 2009).

Event-driven information systems demand a systematic and automatic processing of events. Complex Event Processing (CEP) encompasses methods, techniques, and tools for processing events *while they occur*, i.e., in a continuous and timely fashion. CEP derives valuable higher-level knowledge from lower-level events; this knowledge takes the form of so called complex events, that is, situations that can only be recognized as a combination of several events.

1 Application Areas

Service Oriented Architecture (SOA), Event-Driven Architecture (EDA), cost-reductions in sensor technology and the monitoring of IT systems due to legal, contractual, or operational concerns have lead to a significantly increased generation of events in computer systems in recent years. This development is accompanied by a demand to manage and process these events in an automatic, systematic, and timely fashion. Important application areas for Complex Event Processing (CEP) are the following:

Business Activity Monitoring aims at identifying problems and opportunities in early stages by monitoring business processes and other critical resources. To this end, it summarizes events into so-called key performance indicators such as, e.g., the average run time of a process.

Sensor Networks transmit measured data from the physical world, e.g., to Supervisory Control and Data Acquisition systems that are used for monitoring of industrial facilities. To minimize measurement and other errors, data of multiple sensors must often be combined. Further, higher-level symbolic situations (e.g., fire) must often be derived from raw numerical measurements (e.g., temperature, smoke).

Market data such as stock or commodity prices can also be considered as events. They have to be analyzed in a timely and continuous fashion in order to recognize trends early and react to them automatically, for example, in algorithmic trading.

The situations that must be detected in these applications and their associated information are distributed over several events. They must be derived from several events and their relationships through CEP.

2 Types of Complex Event Processing

The term Complex Event Processing was popularized in [9]; however, CEP has many independent roots in different research fields, including discrete event simulation, active databases, network management, and temporal reasoning. Only in recent years, CEP has emerged as a discipline in its own right and an important trend in industry. The founding of the Event Processing Technical Society [6] in early 2008 underlines this development.

One should distinguish in CEP the case where complex events are specified as a-priori known patters over events and the case where previously unknown patterns should be detected as complex events. In the first case, event query languages offer convenient means to specify complex events and detect them efficiently. In the second case, machine learning and data mining methods are applied to event streams. This article focuses on event query languages since they are the more mature area.

2.1 Relationship to other topics

Detection of complex events is, of course, no an end in itself; an event-driven information system should react automatically and adequately to detected events. Typical reactions include notifications (e.g., to another system or a human user), simple actions (e.g., buy stocks, activate fire-extinguishing installation), or interaction with business processes (e.g., initiation of a new process, cancellation or modification of a running process).

CEP is therefore closely linked with other topics such as visualization of event data for human users, message-based middleware for transport of messages, rule systems for the specification of reactive behavior (e.g., ECA rules and reactive logic programming), and business process management. In this article, however, we only address CEP in the narrower sense of detecting complex events.

3 Event Queries

In contrast to database queries, event queries are evaluated continuously while the events happen. While databases also often work with event-related data (e.g., a history of orders), queries there are one-time and "ad-hoc" against a finite set of data instead of continuous and "standing" against a (conceptually) infinite stream of events as in CEP (cf. Fig. 1).

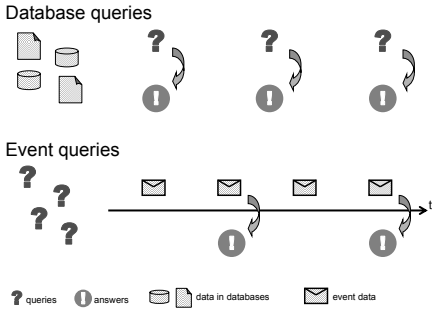

Figure 1: Difference between database and event queries.

The requirements to an event query language can be described with the following four aspects:

Data extraction: Events contain data that is relevant to decide whether and how to react to them. Access to this data must be possible in conditions of queries, in potential reactions, for enrichment with other data (e.g., from database tables), and for construction of new events. Increasingly, events are represented in XML formats; in this case, data can have a quite complex structure.

Composition: It must be possible to join several individual events together, so that their combined occurrences over time yield a complex event. This composition must often be sensitive to data (e.g., join only events concerning the *same* customer).

Temporal Relationships: Event queries often involve temporal conditions expressing that the events must happen within a particular time interval or in a particular order. Other relationships between events, e.g., causality, can also be important.

Accumulation: Queries involving negation (absence of an event) or aggregation of event data are not sensible on infinite streams because they can only be answered correctly when the stream ends. Accordingly, such queries can only be issued against certain finite extracts (or "windows") of a stream, where their result is well-defined.

Additionally, two types of rules are important: **Deductive Rules** define new events based on event queries; they are comparable with views in databases and have no side-effects. We emphasize that these deductive rules operate on events, not on facts (like "traditional" deductive rules from logic programming and deductive databases). **Reactive rules** [3] specify how to react to (complex) events, e.g., with database updates or procedure calls.

4 Prevalent Event Query Languages

Broadly speaking, three different types of languages are currently used to express event queries. In the following, we introduce the core ideas of these three language styles. Additionally, we give an outlook on a recent research project concerned with the design of an event query language at the end of this article. The discussions here are consciously kept short and generalizing; we refer to chapter 3 of [5] for a more detailed elaboration and bibliography.

4.1 Composition Operators

Composition operators have their origins in active database systems [10], though newer systems like Amit [1] run independently from a database. Complex event queries are expressed by composing single events using different composition operators. Typical operators are conjunction of events (all events must happen, possibly at different times), sequence (all events happen in the specified order), and negation within a sequence (an event does not happen in the time between two other events). Nesting of expressions makes it possible to express more complicated queries.

Many language support restrictions on which events should be considered for the composition of a complex event. Event instance selection allows to select, e.g., only the first of last event of a particular type. Event instance consumption prevents the reuse of an event for further complex events if it has already been used in another, earlier complex event.

Composition operators offer a compact and intuitive way to specify complex events, where temporal relationships and negation are well-supported. Event instance selection and consumption is a feature that is not present in the other approaches. Yet, there are sometimes hidden problems with the intuitive understanding of operators, e.g., several variants of the interpretation of a sequence (amongst others, interleaved with other events or not). Further, event data is often being neglected, in particular regarding composition and aggregation.

Currently only very few CEP products are based on composition operators, among them IBM Active Middleware Technology (Amit) and ruleCore.

4.2 Data Stream Query Languages

Data stream query languages like CQL [2] are based on the database query language SQL with the following general idea: data streams, which contain events as tuples, are converted into relations. On these relations a regular SQL query is evaluated. The result (another relation) is then converted back into a data stream. Conceptually, this process is done at every time point of a fixed discrete time axis. (See however [8] for variations.)

For the conversion of streams into relations, window operations such

as “all events of the last hour” or “the last 10 events” are used. For the conversion of the result relation back into a stream there are three options: only tuples that have been added in comparison with the previous result yield a new event, only tuples that have been removed, or simply every tuple of the (current) result.

Data stream query languages are very suitable for aggregation of event data, as particularly necessary for market data, and offer a good integration with databases. Expressing negation and temporal relationships, on the other hand, can often be cumbersome. The conversion from streams to relations and back can be considered somewhat unnatural, as can the prerequisite of a discrete time axis.

SQL-based data stream query languages are currently the most successful approach and are supported in several efficient and scalable industry products. The better known ones include Oracle CEP, Coral8, StreamBase, Aleri and the open-source project Esper. However, there are big differences between the respective variants and important extensions that go beyond the general idea that has been discussed here.

4.3 Production Rules

Production rules, which nowadays are mainly used in business rule management systems like Drools or ILOG JRules, are not an event query language in the narrower sense. The rules are usually tightly couples with a host programming language (e.g., Java) and specify actions to be executed when certain states are entered [3]. The states are expresses as conditions over objects in the so-called working memory, which are also called facts.

Their incremental evaluation (e.g., with Rete) makes production rules also suitable for CEP. Whenever an event occurs, a corresponding fact must be created. Event queries are then expressed as conditions over these facts. In doing so, the programmer has much freedom and little guideline.

CEP with production rules is very flexible and well integrated with existing programming languages. However, it entails working on a lower and —since it is state and not event oriented— somewhat unnatural abstraction level. Especially aggregation and negation are therefore not easy to express. Garbage collection, that is, the removal of events from the working memory, must be programmed manually. (See however [11] for work towards an automatic garbage collection.) Production rules have the reputation to be less efficient than data stream query languages.

Besides their use in business rule management systems that are not focused on events, production rules are also an integral part of the CEP product TIBCO Business Events.

5 CEP in Practical Use and Research

CEP is an industrial growth market as well as an important research area that is emerging from coalescing branches of other research fields. Despite first successful projects in the application areas discussed at the beginning [7, 12], there is still high demand for experiences and comparisons of event query languages in concrete projects. (This is also due to a certain secrecy in algorithmic trading, which is currently still the biggest market for CEP.) Further there are only few benchmarks to compare and predict the performance of CEP systems. Beyond event query languages, reference architectures and design patterns for CEP are of high importance.

5.1 Standardization and Harmonization Activities

Even though the prevalent event query languages can be categorized roughly into three families as done in this article, there are significant differences between the individual languages of a family. Whether a convergence to a single, dominant query language for CEP is possible and advisable is currently in no way agreed upon.

Efforts towards a standard for a SQL-based data stream query language are being underway [8], but not yet within an official standardization body. A standardized XML syntax for production rules is being developed in the framework of the Rule Interchange Format (RIF) by the W3C; however, the special requirements of CEP are not considered there so far. The same applies to the Production Rule Representation (PRR) by the OMG.

To support modeling of events in UML, the OMG has recently issued a request for proposals for an Event Metamodel and Profile (EMP). It explicitly mentions that the EMP should support modeling of CEP functionality.

Activities of the Event Processing Technical Society (EPTS) aim at a coordination and harmonization, amongst others with the work on a glossary of CEP terms and just initiated work on the analysis of event query languages. Further, the EPTS wants to support standardization efforts undertaken by other organizations.

5.2 Outlook on Current Research: XChange$^{\mathbf{EQ}}$

Some of the problems associated with the languages discussed in Chapter 4 can be attributed to the fact that they mix the four aspects of event query languages and, in doing so, neglect some aspects. The research project XChange^{EQ} [4] develops an event query language that follows an approach where queries are expressed in the style of logic formulas and the four aspects are clearly separated. XChange^{EQ} further supports deductive rules over events and direct, pattern-based queries against events in XML formats.

For example, the event query on the right hand side of the following deductive rule expresses that an order with less than 10 items (q) is considered late if it has not been delivered within two days.

$$\begin{array}{lcl} \text{late}(id, t) & \leftarrow & o : \text{order}(id, q),\ s : \text{shipping}(id, t),\ w : \mathsf{extend}(s, 2 \text{ days}), \\ & & \mathsf{while}\ w : \mathsf{not}\ \text{delivery}(t),\ o\ \mathsf{before}\ s,\ q < 10 \end{array}$$

Research on XChange^{EQ} exemplifies the importance of good language design and formal foundations in CEP and tries to rectify problems of the prevalent approaches. The basic idea of writing event queries as logic formulas with a separation of concerns is, e.g., also transferable to production rules and can serve as a guideline for authoring event queries there.

5.3 Further Research Topics

Further research in Complex Event Processing is especially still needed on formal foundations, in particular with regards to expressiveness of languages and optimization. Important for query optimization are the exploitation of shared (sub)expressions of queries (multi query optimization) as well as distributed and parallel evaluation. Further research topics include dealing with uncertainty in events (e.g., with probabilistic methods) and the detection of a-priori unknown complex events (e.g., data mining on event streams).

References

[1] A. Adi and O. Etzion. Amit — the situation manager. *VLDB Journal*, 13(2):177–203, 2004.

[2] A. Arasu, S. Babu, and J. Widom. The CQL continuous query language: Semantic foundations and query execution. *VLDB Journal*, 15(2):121–142, 2006.

[3] B. Berstel, P. Bonnard, F. Bry, M. Eckert, and P.-L. Pătrânjan. Reactive rules on the Web. In *Reasoning Web, Int. Summer School*, number 4636 in LNCS, pages 183–239. Springer, 2007.

[4] F. Bry and M. Eckert. Rule-Based Composite Event Queries: The Language XChange^{EQ} and its Semantics. In *Proc. Int. Conf. on Web Reasoning and Rule Systems*, number 4524 in LNCS, pages 16–30. Springer, 2007.

[5] M. Eckert. *Complex Event Processing with XChange^{EQ}: Language Design, Formal Semantics, and Incremental Evaluation for Querying Events*. PhD thesis, Institute for Informatics, University of Munich, 2008. `http://edoc.ub.uni-muenchen.de/9405/`.

[6] Event Processing Technical Society (EPTS). `http://www.ep-ts.com`.

[7] T. Greiner et al. Business activity monitoring of norisbank taking the example of the application easyCredit and the future adoption of complex event processing (CEP). In *Proc. Int. Symp. on Principles and Practice of Programming in Java*, pages 237–242. ACM, 2006.

[8] N. Jain et al. Towards a streaming SQL standard. In *Proc. Int. Conf. on Very Large Databases*, pages 1379–1390. VLDB Endowment, 2008.

[9] D. C. Luckham. *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems*. Addison-Wesley, 2002.

[10] N. W. Paton, editor. *Active Rules in Database Systems*. Springer, 1998.

[11] K. Walzer, T. Breddin, and M. Groch. Relative temporal constraints in the Rete algorithm for complex event detection. In *Proc. Int. Conf. on Distributed Event-Based Systems*, pages 147–155. ACM, 2008.

[12] G. Wittenburg et al. Fence monitoring — experimental evaluation of a use case for wireless sensor networks. In *Proc. Europ. Conf. on Wireless Sensor Networks*, volume 4373 of *LNCS*, pages 163–178. Springer, 2007.