What's Different About Rules in CEP?

Paul Vincent
TIBCO Software
3303 Hillview Ave, Palo Alto, CA 94304, USA
pvincent@tibco.com
http://tibcoblogs.com/cep http://www.tibco.com

[October Rules Fest 2009]

Abstract

Complex Event Processing can be described as the generic processes and technologies for identifying abstract events. Identifying such events can require several levels of complex event patterns across context, time and sets, and can be defined as a set of "rules" (if <pattern> then <complex event>). Various types of "rule technology" are deployed in CEP, varying from orchestrated SQL-type continuous queries, Event-Condition-Action rules, as well as inference production rules. However, there are various extensions and considerations to be made for event processing over data-oriented rule-based development.

Disclaimer

This is not an academic paper – by "standard" or intended audience. For academic and theoretical aspects of rules in CEP I rely on my academic associates – for example those involved in research in areas like RuleML, the Semantic Web and various logic representations – many of whom I have met in the course of attempting to develop rule standards such as OMG PRR (Production Rule Representation) for UML, and the semantic web extension for rules known as the W3C Rule Interchange Format. Instead, this paper is based on the approach taken by a middleware vendor (TIBCO Software) some 5-6 years ago to apply rule technology (Rete-based inference rules) to the problems of high performance event processing – and in particular complex event processing.

What is Complex Event Processing and why is it interesting?

Let us first start by defining what is meant by "CEP", and how it differs from conventional software paradigms. Traditionally, IT has been involved in automating business processes with ever-increasing complexity and management, mostly around taking input data – which are effectively "events" at the time the data is created – and then storing and organising that data for processing. This data-centric view of the world has served IT well.

On the other hand, traditional data-centric IT has a number of issues (both in development

and run-time costs and agility) and does not really advance the state of the art from the Edwardian-era office, with its equivalent card record index, vacuum message tubes or message boys, and silo'd, specialist view of processes and services. Of course, a variety of IT techniques have evolved to progress the art of software engineering, such as BPM and SOA, decision services automating business rules, and model-driven engineering (via UML). Less common (and still to be proven) are areas like RDF (schema-flexible) data stores and other "semantic web" technologies.

Meanwhile, in real-life, there has been an increasing rate of "input" or "information arriving" for processing (or potential processing). Processing these events via the data-based approach has not always been found to be performant or attainable. Examples of systems with large event rates are RFID handling in the supply chain, monitoring other business and IT services, (packet-switched era) telecommunications, and of course algorithmic trading.

The increase in "event rates" has in turn stressed the software engineering practices used for developing systems, regardless of architecture. Agility is always desirable in software engineering, but difficult to achieve whilst maintaining reasonable or high performance levels. But for event operations in areas such as Business Activity Monitoring (BAM) and Real-Time Business Intelligence (sometimes termed "Operational Intelligence") there may be a wide variety of different event combinations that I need to identify on an ad-hoc, after-install basis.

The need for flexible, high performance event processing has led to the development of what Luckham calls "complex event processing" – technologies to specify abstract or aggregate events that provide specific business information in a timely manner. CEP covers problems like "situation awareness", "sense and respond", and "track and trace". As the numbers of information channels multiply, and the volumes of events on those channels increase, the practice of "dump to database *then* process" becomes less desirable or even feasible; however the desire to increasingly automate tasks for Operational Intelligence (providing information and intelligent processing of real-time business data) is becoming increasingly interesting to businesses.

How do rules play in CEP?

Now let us define what we mean by "rule". For attendees at ORF, a "rule" will pretty much mean a production rule in an inference engine. To a constraint programming expert, it means a potentially relaxable constraint on some business entity (or a constraint statement over several entities). To a database designer, it means the entity relationship rules that tie data concepts together.

Mostly in event processing, rules are processing operations that are made on events. In complex event processing, these operations are used to create aggregate in intermediate events – even an operation such as the "total number of events" is considered a "complex event" (although whether this is especially interesting is another matter).

There are multiple ways rules apply in CEP, therefore.

1. There are relationship rules for the underlying events and supporting data models – the static rules that are usually

hard-coded as the Business Object Model or Database schema (although in the semantic web world there is flexibility provided here through RDF and RDFS techniques through to OWL ontologies).

2. There are the rules relating to definition of event patterns. These can take many forms:
    a. Rules that determine state changes in entities such as incomplete complex events
    b. Rules that identify some pattern of events and data and deduce some new fact (as a new event or data update).

For the most part we are interested in the latter.

What sorts of rule types or patterns do we have in CEP?

So far the reader will probably be wondering if there is any particular difference between the rules a standard Business Rule Engine (BRE) provides and the ones used in CEP, especially if one considers an event as a subtype of data. However, one of the characteristics of "complex events" is that the complex aggregation can be time-dependent – for example, a complex event e could be derived from event "a" occurring followed by an event "b". This means that complex event pattern matching needs to involve state: event "e" is constructed if I'm in the state of having observed event "a", and am awaiting event "b". It should therefore be obvious that one of the first representations of rules useful in CEP is the state transition rule, including time-out rules, and the associated state model metaphor and state machine semantics. In this way a complex event can be defined in terms of multiple intermediate states and their associated state transition rules.

```
If e is in state s and event e'
occurs [and other conditions]
then transition e to state s'
```

Note that a state machine is readily mappable to a rules engine: states are simply properties of entities, and state transition rules are the rules for setting this state property, with a compulsory condition being the prior value of the entity's state...

State transition rules are of course merely a specialization of the common-or-garden event-condition-action (ECA) rule, which is also very relevant in simpler (or orchestrated) complex event production:

```
If event e' occurs [and other
conditions]
then transform e to e'
```

Of course, such ECA rules are themselves specialisations of our everyday production rules, which in an inference engine can be triggered by other rule firings and by themselves can construct some complex event:

```
If [data conditions]
then construct event e
```

Note though we have a small problem with event definition here: events are usually described as immutable: they represent "observations" whose metadata is unalterable post-occurance. Yet to construct a "complex event" means operations "on" an event that can eventually be published. Usually this means that the term "event object" is used to describe something that will end up as a published

outgoing event or as a “constructed” or “deduced” observation.

Other representations of temporal rules?

Although an explicit state representation is one of the easier ways to represent time and state in an event –driven situation, there are other approaches to representing “time” and “time periods” in rule conditions. Some of the solutions are:

a. Represent the rule as some process, with process steps representing temporal conditions between event occurences and operations. Clearly any individual event can participate in multiple such processes. For example, consider the process model notation BPMN with event inputs driving the flow / orchestration of any particular process.
b. Represent the rule as some aggregation query to which a time policy is attached. This is usually represented as some kind of SQL-type query such as “over time window t”, and is particularly useful for dealing with “streams” of events where events need mostly to be considered with their adjacent events within the same channel.
c. Represent temporal conditions as rule conditions. Of course most rule languages can handle timestamps and time comparisons, so this is the de-facto event processing behaviour in a rule engine.
d. Represent the event pattern as a kind of regular expression including time statements or constraints.

Probably there are more to come!

CEP rule development challenges

Complex event patterns include one other representational ”problem” in addition to temporal rules (and arbitrarily complex event collections): the problem of representing a “missing” event. In a Rete type production rule one matches events and data. But how do you represent an event that needs to NOT have occurred (within a particular time)? Examples are:

i. State model representing the intermediate complex event: if the unwanted event occurs within a particular time then the complex event is invalidated.
ii. Multiple rules representing the event pattern construction, with an “invalidating rule” if the unwanted event occurs.

By far the biggest challenge with event processing remains the need for high performance across high volumes of information in a reliable failsafe manner with minimal latency. For large scale processing problems this means multiprocessing and even distributed processing, with associated architectures such as grids, distributed caches, distributed rules, failover multi-threaded rule engines in hot standby, and so forth. For the rule designer, this means careful attention to the layout and runtime cost of rules to ensure that events are not queued or lost due to excessive rule process times.

Other Event-driven Considerations

The best way for regular rule-programmers to understand the changes related to event-based

rule processing is to consider an example. Important considerations are:

1. Events can be defined in terms of metadata, payload and source. However, the main programming consideration is that they are transient. So an event will often have an automatic expiry – a Time To Live. Of course, you can specify an event as existing forever until you decided to delete it yourself, but the more common models are to
    - Process the event immediately and then let it expire / be removed from memory
    - Keep the event around for a specified time to handle all known temporal event patterns
    .
2. The rule engine "Run To Completion" cycle needs to match the event load for any particular rule engine instance. In other words, its no use running a Miss Manners-type rule exercise while events are arriving! The normal solution for this is to distribute rule processing across multiple engines or threads to ensure that events can still be consumed at the appropriate rate. (And of course the rule approach should handle events like "new guest" in the case of Miss Manners!).

3. Performance in rule-based systems is, like everywhere else, a function of work done. While the basics of Rete-based rules provides efficiencies in some scales, this does not alter the fact that minimizing work done (and understanding the algorithms in use to allow this) can be critically important in tuning a system to handle the maximum throughput while applying some business logic. Areas to consider are:
    - The numbers of events (and event objects) in working memory at any one time. If the object isn't relevant to the current rule processing cycle, should it be in a different rule engine instance? Or use a different rulevariable and object type?
    - The number of rules and rule conditions (mapping to filters and joins) and any ordering considerations.
    - The need to organise the rules to meet the required manageability objectives.

    In addition, the various techniques for multi-agent and distributed systems apply when such techniques are used for CEP.

Summary

The use of rule-based programming techniques is applicable to event processing as it is to data processing. Apart from dealing with state management (including memory management for stateful processing) and event management (when and how to process and dispose of events), the event-driven rule system for CEP should hold no worries for the skilled rule developer.

Suggested Reading

The Power of Events, David Luckham: also http://complexevents.com

Event Processing Technical Society glossary of terms: http://www.ep-ts.com

TIBCO Complex Event Processing blog: http://tibcoblogs.com/cep

TIBCO BusinessEvents Technical White Paper by request from the author or info@tibco.com