Blending Complex Event Processing with the RETE Algorithm

Kay-Uwe Schmidt[1], Roland Stühmer[1], and Ljiljana Stojanovic[2]

[1] SAP Research, CEC Karlsruhe
Vincenz-Prießnitz-Straße 1, 76131 Karlsruhe
{kay-uwe.schmidt,roland.stuehmer}@sap.com
http://www.sap.com
[2] FZI Forschungszentrum Informatik
Haid-und-Neu-Straße 10-14, 76131 Karlsruhe
ljiljana.stojanovic@fzi.de
http://www.fzi.de

Abstract. Modern event-driven applications implement sophisticated and highly specialized algorithms for detecting and correlating events from event streams or clouds. The correlation logic and resulting actions are declaratively defined as EA (event action) rules. An EA language allows the definition of complex correlation rules with the help of logical, temporal, content-based and other operators. On the other hand, production rule systems provide a declarative means to express and compute CA (condition action) rules. CA rules define constraints. The most prominent scientific and commercial production rule systems rely on the RETE algorithm in order to efficiently compute CA rules. As the industry is on the way towards a loosely coupled service-oriented and event-driven architecture, there is a high demand for an integrated solution of computing ECA (event condition action) rules. In this paper we give a survey of already finished as well as ongoing research in the field of combining event processing algorithms with the RETE algorithm. We compare the different approaches and present our approach of combining SnoopIB, a specialized event detection algorithm, with the RETE algorithm.

Key words: RETE Algorithm, CEP, Event, ECA, Rules

1 Introduction

In a production rule system the RETE algorithm [1, 2] or derivatives thereof like TREAT [3], LEAPS [4] or GATOR [5] are used to establish which rules must be fired. The RETE algorithm is the execution kernel of the rule-based language, OPS5 [6]. OPS5 is a condition action (CA) production rule language. The RETE algorithm is concerned with the condition parts of rules. When matching rules are found, their action parts are executed. An extension to CA rules are event condition action (ECA) rules, comprised of an additional event part. This kind of rules specify an action which is executed upon the detection of an event but only if the condition is fulfilled at the same time. Both the event part and the condition part of a rule may consist of complex patterns which must be fulfilled for the rule to fire. The event pattern must be detected from one or more streams of events and the condition pattern must be matched from among business objects (facts) pertaining to the application state.

The open research question remains of how both matching operations can be integrated in an efficient manner. There are many approaches of either adding event processing to production rule systems or integrating scalable condition pattern matching algorithms into event processing engines. In this paper we describe the different approaches and present their Pros and Cons. We present a novel approach of combining the two rule execution paradigms: Data-driven and event-driven execution. Our algorithm is an excerpt of our ongoing research of adding complex event processing (CEP) to RETE. It is a middle course which refrains from merging the two algorithms but still exposes synergistic effects. However, the question is not whether the event and condition specifications should be consolidated. Conceptually, events and conditions are concerned with different information with different semantics. Therefore, it makes sense to distinguish events from non-temporal facts in an ECA rule specification. Considering an application as a state machine, events represent transitions, cf. [7]. When an event is detected, the appropriate transition is taken. The event may then be discarded. Facts, on the other hand, represent state. A rule condition determines a state to be held before a transition may be taken. Facts remain valid and must be retained until they are explicitly removed in a state change. The analogy of a state machine provides a usable guide to distinguish between events and facts. For the implementation of an ECA rule engine, on the other hand, there is no obligation to keep events and facts separated. However, as events are transient, they must efficiently be discarded after they cannot be used any further.

Table 1 outlines some notable differences in semantics and behavior of events and non-temporal facts. Events are transient and are consumed after they are delivered to all consumers. Facts, on the other hand, are persistent and must explicitly be deleted. Events are usually immutable, whereas facts may be changed. Analogies may be drawn to transitions and state, respectively. Another analogy from [8] compares facts to (computer-)written text. Text may be altered or deleted. Also it does not expire by itself. It is available for anyone to retrieve it (i.e. pull). Events, on the other hand, are like the spoken word. It is available only to the people who listen at the right time (i.e. pushed to subscribers). Also the spoken word is immutable.

The paper is structured as follows: Section 2 surveys the different approaches of adding event processing capabilities to the RETE algorithm. In Section 3 the several approaches introduced in Section 2 are accessed and the pros and cons are outlined. We give an elaborated insight into our algorithm that combines CEP with RETE in Section 4. We conclude the paper in Section 5 with a discussion and outlook.

Table 1. Events and non-temporal facts: Events are not like other facts which RETE usually handles. This table briefly outlines some notable differences in semantics and behavior.

Events	non-temporal facts
transient	persistent
usually not modifiable	modifiable
Analogy:	
transition	state
spoken word	written text
Behavior:	
received via *push*	fetched via *pull*

2 A Survey of Approaches Adding Event Processing Capabilities to the RETE Algorithm

Maloof et al. [9] extend RETE to allow reasoning about relative time representations. This is done by introducing temporal operators *before*, *during* and *after* which may be used to compare temporal facts from the working memory. Events are seen as having a duration, therefore an interval-based semantics is assumed. However it is not clearly pointed out how exactly the operators are evaluated and how the interval start or end are used.

Berstel [10] also proposes an extension to the RETE algorithm to enable the relative operators *before* and *after*. Only time-point semantics is used for events. This exposes unwanted effects for the correct detection of sequences when handling non-instantaneous events. Two extensive intervals, which mostly overlap, will be detected as occurring one after the other, because only their end points are considered for comparison. Nesting of sequence operators provides even more unexpected results when transitivity is concerned.

Berstel also devised a garbage-collection mechanism to expunge unneeded events when they are no longer needed. Events are special facts which are asserted through a special API. Only these facts are considered for garbage-collection. When a join node encounters a new event in one of its input memories, the lifetime of the event is calculated from the time stamp of the event and the join conditions of the node which must at least be fulfilled. A callback service is used by the join node to raise a timeout when the lifetime is passed. The node then issues its parent memory to delete its local reference to the event. Global retraction of events is not discussed.

Walzer [11, 12] extended the RETE algorithm with the 13 temporal relations from Allen [13]. They cover all relationships intervals can have. Accordingly, Walzer uses interval-based semantics for events. Temporal operators for each of the 13 relations are introduced, e.g. *equal*, *before*, *after*, *during*, *overlaps*, etc.

The operators can be quantified, e.g. specifying how much two intervals overlap in the beginning and in the end. Ranges for these quantifiers are also allowed.

Walzer implemented a garbage-collection mechanism, taking into account the constraints in join nodes, time-based windows in join nodes, the time stamps of events and maximum lifetimes of events. When a new token or a new working memory element (WME) reaches a RETE memory, their lifetimes are calculated if they contain events. The working memory is notified to increase the reference counts on these objects. Using the calculated lifetimes the working memory starts the creation of timers in the garbage-collector. When a timer runs out, the garbage-collector removes the referenced token or WME from its memory. The reference count in the working memory is then decreased, and if it reaches zero, the event is completely retracted from the working memory.

3 Comparing the Different Approaches

There are several features concerning CEP in which the mentioned approaches differ and there are some which they have in common due to the RETE algorithm.

An important distinctive feature is the treatment of events as time intervals instead of time points. For the field of CEP this has been discussed for quite some time, e.g. in [14]. To receive expected results from temporal operators, events must be represented as intervals if they are detected over a period of time.

The next distinctive feature is garbage-collection. Events arrive in streams. They may then be processed and must be discarded in order not to use indefinite amounts of space. Events are transient representations of incidents which happened. These events may be used on a subscriber-basis by different consumers, i.e. *rules* in the RETE context. When no further consumer is interested in an event, it can be discarded. Thus, there is a fundamental difference between facts and events. Facts have a potentially indefinite lifetime. They are only removed if they are explicitly retracted. Events, on the other hand, have individual lifetimes which may be calculated from constraints imposed by their consumers. After these lifetimes have expired, the event can be completely removed.

Another distinctive feature is the operators introduced by the different approaches. Berstel proposes the operators *before* and *after* for time-points. As there are no intervals, Berstel's approach does not contain an *during*-operator. Maloof adds the *during*-operator. Walzer finally adds all operators representing the 13 interval relations. This is the richest set of operators one of the approaches offers.

All approaches share the notion of a working memory. All events go into the working memory and stay there as long as they are not garbage-collected. However, providing a global pool of events is not appropriate for a publish/subscribe environment. For example an event should be only visible to rules which subscribe to it and there should be no reason to expose the event further. The working memory, however, retains the event as long as it might still match one of the current rules. Only then is the event discarded. When a new rule is introduced it should not match events from the past; Ideally all events will be deleted. However, a new rule might still see an older event if is still in the working memory, retained by some existing rule / consumer. Therefore the matches for the new rule depend on the existence of other rules. This is a very much undesired effect for a rule system.

The semantics of an event being visible in the working memory is therefore unspecific, meaning only that the event has not been consumed by all subscribers. Other features of the working memory are also not applicable for events. This questions the purpose of adding events to a working memory in the first place. For example as events should be immutable (cf. *spoken word* analogy mentioned in Section 1) the **modify**-operation of the working memory is not applicable. In fact, Walzer marks events as immutable facts (cf. [15]) in her implementation for the rule engine Drools[3] [16, 17]. On top of that, **retract**-functionality seems unreasonable for events, as well. An event either happened or it did not happen. All subscribers should consistently receive it.

None of the RETE-based approaches have a notion of event selection and consumption. Selection and consumption are described e.g. in [18] as so-called *contexts*. For each event operator they define which events from each input should be used for correlation, if there are more than one, and whether a used event can be part of further matches or is consumed.

In all of the approaches mentioned above a given join node correlates the available events from both inputs. This results in a cross product of all events which have a suitable lifetime and therefore are not garbage-collected yet. However, there is no way to specify e.g. that successive events should supersede their predecessors as in the so-called Recent context. Also, there is no way to consume events in the process of correlation. Rather, events will take part in further correlations as long as only its temporal constraints are fulfilled.

4 Enhancing RETE with SnoopIB

Instead of incorporating the event detection in the RETE network, we decided to process events in a disjoint set of nodes and only merge the two graphs at the final nodes. Figure 1 shows an example graph.

This allows us to retain an event graph which is not subject to assumptions from the RETE algorithm. We use an event detection graph as proposed with the event pattern language Snoop [18], extended with interval-based semantics for events [19]. Such an event detection graph is a DAG, constructed as follows: All simple events are at the bottom. There is one simple event node per simple event type. Complex event expressions are at the top of the graph. Sub-expressions form the inner nodes, down to the indivisible, i.e. *simple*, nodes at the bottom. Common sub-expressions shared by more than one rule result in node sharing.

Simple events coming from the event sources are fed into the graph at the bottom. According to the connected nodes, the events are either discarded, queued or propagated further if they match an event expression.

[3] `http://www.jboss.org/drools`

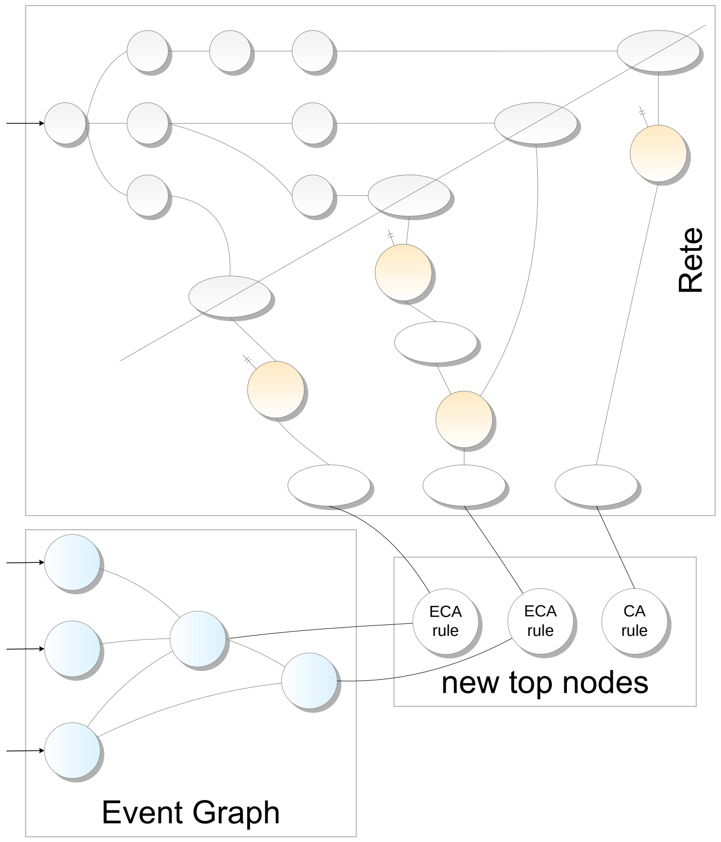

Fig. 1. Coupling RETE with CEP. The RETE network, depicted at the top, is responsible for rule conditions and finds matching facts fulfilling rule conditions. The event detection graph, at bottom left, is responsible for event specifications and finds events matching an event pattern. Both RETE and the event graph propagate their results to newly introduced top nodes. These nodes trigger rule actions.

The RETE network and the described event detection graph have several things in common. They are both DAGs used for incremental pattern matching, in a data-driven, bottom-up fashion. Both employ state saving by not recalculating previous matches. Both reuse partial matches shared by more than one rule. However, especially for the application of CEP, the event graph has some advantages.

- The event graph can have nodes with more than two input edges. This more closely resembles higher-level event operators from Snoop which we are using.
- Also garbage-collection comes more naturally. Events are stored and discarded in the queues of event nodes where they are still needed. There is

no global repository for unused or partially unused events like the working memory from RETE.
- Consumption modes are selectable (on a per-node basis, if necessary). This allows for the controlled selection of events from a stream, if more than one events have arrived and might be fitting for a match. For example in selected applications only the newest event of its kind might be of interest. This behavior is called *Recent context*. There are several contexts as described in [18].

On top of the event graph and RETE a new layer of nodes is introduced, cf. Figure 1 at bottom right. It represents the rules. For a CA rule, the production node from RETE is connected solely to the rule node. For an ECA rule, additionally an event node is connected to the rule node. The rule node fires its associated rule actions according to the ECA semantics. For an ECA rule action to fire, an event must be detected, and for its complete interval, the condition must be fulfilled.

Vice versa this means that no events need to be correlated as long as RETE supplies no matched tokens. Using this realization, computation time (and space) can be saved in the event detection graph, when unneeded event nodes are disabled during the time the rule condition has no matches. We are disabling nodes recursively down from the top event node (which is to-be-disabled) down to all of its contributing children nodes. Care must be taken when encountering a shared node which has further consumers. Such a node is only disabled when all its consumers are disabled.

The achieved savings from this procedure varies depending on several factors. If no event nodes are reused, the saved computations are a function of the event frequencies and the probability of the rule condition being unmatched. To calculate the necessary event operations of one rule, one must consider the added frequencies of all participating events, simple and complex, including all intermediate stages.

However, determining the average frequency of events might not always be practicable or accurate, e.g. if a source emits events aperiodically. Also, if nodes are shared by several consumers, assessing the amount of saved computations becomes more difficult. Event operations can only be cut down if all consumers are disabled. The event-frequency of each participating node must then be multiplied by the probability of all other consumers being disabled as well.

5 Conclusions and Outlook

In this paper we compared different approaches of incorporating CEP in a RETE network. As RETE is designed to reason over persistent and mutable facts, it is not a natural fit for doing event processing. Many changes towards temporal capabilities as well as event expiry must be added. And because the RETE network relies on a working memory where events reside, such an approach does not resemble an *event processing network* as described in [20]. In such a network events are propagated forward[4] and are only visible to nodes which subscribe to the events and have not yet consumed them.

We presented our approach of blending CEP with RETE as a solution to avoid the indicated mismatch of requirements. Our approach works by combining dedicated event processing nodes with RETE nodes. In our opinion it is a promising way of coupling event and condition processing, creating synergistic effects without constricting the features of CEP. Features of CEP which are not available in current RETE adaptation are mentioned in this paper. The features include selectable detection contexts to determine the precise candidates of events for correlation. Also the features include local visibility of events and garbage collection which guarantees the isolation of different rules. These requirements for CEP have not been addressed by the current RETE-based adaptations.

As an outlook for our approach we are going to evaluate our implementation, in order to produce numbers for the savings in event processing time mentioned in Section 4. Benchmarking ECA rule performance depends on many factors and we are looking for a suitable ratio of event frequency, number of rules, and changes to the working memory. For that purpose we are going to evaluate existing metrics on scoring of event-driven and rule-based systems.

In the future we are also looking into further improvements for our approach. The four detection contexts from Snoop can possibly be generalized by an algebra for event selection and consumption as proposed in [22] or [23]. Using these event algebras, one is independent of predefined detection contexts. Also we are exploring the use of advanced RETE derivatives like TREAT, LEAPS or GATOR as the preferred discrimination network to satisfy rule conditions and to combine it with CEP.

References

1. Charles L. Forgy. *On the efficient implementation of production systems.* PhD thesis, Carnegie Mellon University, Pittsburgh, PA, USA, 1979.
2. Charles L. Forgy. Rete: a fast algorithm for the many pattern/many object pattern match problem. *Artificial Intelligence*, 19:17–37, 1982.
3. Daniel P. Miranker. Treat: A better match algorithm for ai production system matching. In *AAAI*, pages 42–47, 1987.
4. Don Batory. The leaps algorithms. Technical report, University of Texas at Austin, Austin, TX, USA, 1994.
5. E. N. Hanson and M. S. Hasan. Gator: An optimized discrimination network for active database rule condition testing, technical report tr-93-036. Technical report, CIS Department, University of Florida, 1993.
6. Charles L. Forgy. The ops5 user's manual. technical report cmu-cs-81-135, 1981.

[4] About propagation and visibility: Strictly speaking, according to the Event Processing Glossary [21], an event processing network *may* have feedback arcs, i.e. cycles. However, events are still confined in their visibility to only those nodes which can still use them.

7. Opher Etzion. On events and data. Online Article. `http://epthinking.blogspot.com/2008/07/on-events-and-data.html`, July 2008.
8. François Bry and Michael Eckert. Twelve theses on reactive rules for the web. In Torsten Grust, Hagen Höpfner, Arantza Illarramendi, Stefan Jablonski, Marco Mesiti, Sascha Müller, Paula-Lavinia Patranjan, Kai-Uwe Sattler, Myra Spiliopoulou, and Jef Wijsen, editors, *EDBT Workshops*, volume 4254 of *Lecture Notes in Computer Science*, pages 842–854. Springer, 2006.
9. M.A. Maloof and K.J. Kochut. Modifying rete to reason temporally. *Tools with Artificial Intelligence, 1993. TAI '93. Proceedings., Fifth International Conference on*, pages 472–473, Nov 1993.
10. Bruno Berstel. Extending the rete algorithm for event management. In *Proc. Ninth International Symposium on Temporal Representation and Reasoning TIME 2002*, pages 49–51, Washington, DC, USA, 7–9 July 2002. IEEE Computer Society.
11. Karen Walzer, Alexander Schill, and Alexander Löser. Temporal constraints for rule-based event processing. In Aparna S. Varde and Jian Pei, editors, *PIKM*, pages 93–100. ACM, 2007.
12. Karen Walzer, Tino Breddin, and Matthias Groch. Relative temporal constraints in the rete algorithm for complex event detection. In *DEBS '08: Proceedings of the second international conference on Distributed event-based systems*, pages 147–155, New York, NY, USA, 2008. ACM.
13. James F. Allen. Maintaining knowledge about temporal intervals. *Commun. ACM*, 26(11):832–843, 1983.
14. Antony Galton and Juan Carlos Augusto. Two approaches to event definition. In *DEXA '02: Proceedings of the 13th International Conference on Database and Expert Systems Applications*, pages 547–556, London, UK, 2002. Springer-Verlag.
15. Michael Neale. Drools 5.0 m1 - new and noteworthy. Online Article. `http://blog.athico.com/2008/07/drools-50-m1-new-and-noteworthy.html`, July 2008.
16. Mark Proctor. Relational declarative programming with jboss drools. In *SYNASC '07: Proceedings of the Ninth International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*, page 5, Washington, DC, USA, 2007. IEEE Computer Society.
17. Mark Proctor, Michael Neale, Bob McWhirter, Kris Verlaenen, Edson Tirelli, Alexander Bagerman, Michael Frandsen, Fernando Meyer, Geoffrey De Smet, Toni Rikkola, Steven Williams, and Ben Truit. Drools, 2007.
18. Sharma Chakravarthy, V. Krishnaprasad, Eman Anwar, and S. K. Kim. Composite events for active databases: Semantics, contexts and detection. In Jorge B. Bocca, Matthias Jarke, and Carlo Zaniolo, editors, *20th International Conference on Very Large Data Bases, September 12–15, 1994, Santiago, Chile proceedings*, pages 606–617, Los Altos, CA 94022, USA, 1994. Morgan Kaufmann Publishers.
19. Raman Adaikkalavan and Sharma Chakravarthy. Snoopib: Interval-based event specification and detection for active databases. *Data Knowl. Eng.*, 59(1):139–165, 2006.
20. David C. Luckham. *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2001.
21. David C. Luckham and Roy Schulte. Event processing glossary 2008. Online Resource. `http://complexevents.com/?p=362`, May 2008.
22. D. Zimmer and R. Unland. On the semantics of complex events in active database management systems. In *ICDE '99: Proceedings of the 15th International Conference on Data Engineering*, pages 392–399, Washington, DC, USA, Mar 1999. IEEE Computer Society.

23. A. Hinze and A. Voisard. A parameterized algebra for event notification services. *Ninth International Symposium on Temporal Representation and Reasoning, 2002. TIME 2002*, pages 61–63, 2002.