EVENT LIFETIME CALCULATION BASED ON TEMPORAL RELATIONSHIPS

Karen Walzer, Thomas Heinze and Anja Klein*
SAP Research, CEC Dresden, Chemnitzer Strasse, Dresden, Germany

Keywords: Rete algorithm, Rule-based systems, Event processing, Garbage collection.

Abstract: In the last decade, the detection and processing of complex events has increasingly gained importance, since it allows an immediate reaction to changes to instantly adapt business processes to varying situations. However, currently there is a discrepancy in tools available for modelling constraints on business processes, mainly rule-based systems, and those providing complex event processing.
This paper deals with the challenge of identifying unneeded events in rule-based systems in order to delete them to save memory to enable event processing. We adapt an existing approach of lifetime calculation to suit interval-based semantics for event timestamps. In a subsequent validation of our concepts, we were able to demonstrate that they lead to improved event processing capabilities compared to a traditional rule-based system.

1 INTRODUCTION

Whereas rule-based systems are currently used to model integrity constraints and conditions on business processes, they are not used to define and detect complex events. Instead complex event processing systems are used for this purpose. As a result, it is preferable that rule-based systems used to define business processes also facilitate the definition and processing of complex events.

An event represents a change in state of a certain object of the world, it is an "occurrence of significance in a system" (Luckham, 2002). Complex event processing performs a set of operations on event data in order to detect certain logical, temporal or causal combinations of single events in this data. The complex events of interest are defined in a description language and detected by a pattern matching mechanism such as finite state automata (Sánchez et al., 2005). On an occurrence of a specified event pattern, an action such as the creation of a complex event is performed. Rule-based systems (RBS) describe knowledge in the form of facts and if-then rules. A rule expresses that *if* a condition is satisfied, *then* an action is executed. The Rete algorithm (Forgy, 1982) is commonly used in rule-based systems to perform pattern matching, matching a set of facts against a set of rules. It does not delete events or partial results and has no means of determining when events can be discarded besides a manual deletion. Since events are not manually deleted, this is unsuitable for complex event processing. This paper addresses this problem by making the following contributions:

- We define when an event can be considered as being *unneeded* and how this influences its lifetime.
- We extend an existing lifetime calculation (Teodosiu and Pollak, 1992) to suit interval-based timestamp semantics.

The next section presents our approach to determine the lifetime of an event by its constraint satisfiability. After this, related work is outlined. In Sect. 4, we discuss our evaluation. Finally Sect. 5 summarizes our work.

2 LIFETIME CALCULATION

In this section, after giving same basic definitions, the our temporal operators and the lifetime calculation (Teodosiu and Pollak, 1992) we base our work on are briefly introduced. Then, our extension of this calculation for interval-based timestamp semantics is described.

*The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement n° 224282.

Walzer K., Heinze T. and Klein A. (2009).
EVENT LIFETIME CALCULATION BASED ON TEMPORAL RELATIONSHIPS.
In *Proceedings of the International Conference on Knowledge Engineering and Ontology Development*, pages 269-274
DOI: 10.5220/0002305502690274

2.1 Definitions

We will define the terms event, instantaneous and complex event, needed and unneeded event in the following as a foundation of our work.

Let $\mathbb{D}$ be a set of atomic values where atomic values are primitive or complex data types, such as strings or records. Let $\mathbb{T} = (T; \leq)$ be an ordered time domain. Then, let $\mathbb{I} := \{[t_s, t_e] \in T \times T | t_s \leq t_e\}$ be a set of time intervals with t_s as start and t_e as the end time-point of the interval. Then, let a *set of events* be defined as $\mathbb{E} := \{(k_1, .., k_n, t_s, t_e) | k_i \in \mathbb{D}, [t_s, t_e] \in \mathbb{I}\}$. That means we use interval-based timestamp semantics for events. Events can have a duration. In contrast, when using time-point semantics events only occur in a time-point.

Instantaneous events are events with a duration of zero, i. e. $t_s = t_e$. *Complex events* are abstractions of single or complex events. Let $\mathbb{C} := \{(e_1, .., e_n, e_{n+1},) | e_i \in \mathbb{E}, \mathbb{C} \subset \mathbb{E}$ be the *set of complex events*. They can have a duration greater zero, i. e. $t_s \leq t_e$. We define an event to be *needed* as long as it can contribute to any matches in the future. All other events are *unneeded*. An event's *lifetime* τ states the time that an event is needed. An event only needs to be stored for its lifetime. If it is stored longer, it does not contribute to any further matches, but just blocks memory.

2.2 Temporal Operators in Rete

We base our temporal operators on the work of Allen (Allen, 1983) who defined thirteen possible operators to describe the relationships between two time intervals. We enhanced these operators by allowing an operator-specific number of parameters denoting quantitative time constraints. Table 1 states the operators for events e_i and e_j (first column) and their possible parameter constellations (top) with the interpretation of the parameters (table entries). The function $d(t_1, t_2) = t_2 - t_1$ denotes the difference of two timestamps t_1 and t_2.

2.3 Teodosiu and Pollak's Calculation

Teodosiu and Pollak (Teodosiu and Pollak, 1992) allow the specification of time intervals $[\alpha, \beta]$ that define the time an event should occur with respect to another event. For instance, e_2 in $[e_1 + \alpha, e_1 + \beta]$ denotes that event e_2 should occur between α and β time units after event e_1. Negative values for α or β denote that an event should occur a specified amount of time units before the other event. By calculating the dependency matrix of an event's temporal constraints, it is possible to determine its lifetime. They define a *temporal distance* d_{e_i,e_j} as the range of possible values for the difference $t_{e_j} - t_{e_i}$ between the absolute arrival times of two events e_i and e_j. These intervals can be entered in a matrix whose closure leads to a temporal dependency matrix $D_0^* \in \mathbb{R}^2_{n \times n}$ which can be used to calculate an event's lifetime.

2.4 Extension to Intervals

Since we use interval-based semantics for event timestamp definition, we distinguish two cases according to what information is available to us: 1) either events arrive at their start time-point, e.g. representing the initiation of processes of a certain duration or 2) they arrive at their end time-point in our RBS. In the first case, they can be discarded once they do not meet their start time-point constraints. In the second case, constraints concerning its end time-point need to be considered .

Next, we provide the calculations of an event's lifetime for Allen's operators for each of these two cases.

2.4.1 Bounds for Allen's Operators

Allen's operators define constraints on the relation of the start and end time-points of two events. We define $d^{ls}_{e_i,e_j}$ to be a lower bound on an event start time-point, $d^{us}_{e_i,e_j}$ an upper bound on a start time-point, $d^{le}_{e_i,e_j}$ a lower bound on an end time-point, and $d^{ue}_{e_i,e_j}$ an upper bound on an end time-point. In the following, we explain how the upper and lower bounds can be determined for the relative temporal operators.

Events Arriving at their Start Time-point. Using the operators, the upper bounds $d^{ue}_{e_i,e_j}$ of the temporal distance of an event e_i's start time-point used in an operator with event e_j, both events arriving at their start time-points, can be estimated as follows based on their temporal constraints:

$$d^{us}_{e_i,e_j} = \begin{cases} 0 & \text{if } t_{s_{e_i}} = t_{s_{e_j}} \\ -\infty & \text{if } t_{s_{e_i}} > t_{s_{e_j}} \\ \infty & \text{otherwise} \end{cases} \quad (1)$$

That means, if the start time $t_{s_{e_i}}$ of an event e_i is defined to be greater than the start time $t_{s_{e_j}}$ of e_j as e. g. for the *during* operator, then event e_j should have already be arrived at an indeterminate time-point in the past to lead to a match with e_i. Thus, the upper bound of e_i, e_j is $-\infty$, after matching with all existing events, event e_i does not have to be kept, since no future matches can take place. If both events e_i and e_j are

Table 1: Allen's operators (Ops) and their parametrization (P).

Ops \ P	Allen's Definition	a	a,b	$a1,a2,b1,b2$
meets	$d(t_{e_{e_i}},t_{s_{e_j}})=0 \wedge$ $d(t_{s_{e_i}},t_{s_{e_j}})>0$	$0<d(t_{e_{e_i}},t_{s_{e_j}})\leq a$	-	-
met-by	$d(t_{e_{e_j}},t_{s_{e_i}})=0 \wedge$ $d(t_{s_{e_j}},t_{s_{e_i}})>0$	$0<d(t_{e_{e_j}},t_{s_{e_i}})\leq a$	-	-
equals	$d(t_{s_{e_i}},t_{s_{e_j}})=0 \wedge$ $d(t_{e_{e_i}},t_{e_{e_j}})=0$	$\lvert d(t_{s_{e_i}},t_{s_{e_j}})\rvert \leq a \wedge$ $d(t_{e_{e_i}},t_{e_{e_j}})\leq a$	$\lvert d(t_{s_{e_i}},t_{s_{e_j}})\rvert \leq a \wedge$ $d(t_{e_{e_i}},t_{e_{e_j}})\leq b$	-
before	$0<d(t_{e_{e_i}},t_{s_{e_j}})\leq\infty$	$d(t_{e_{e_i}},t_{s_{e_j}})=a$	$a\leq d(t_{e_{e_i}},t_{s_{e_j}})\leq b$	-
after	$0<d(t_{e_{e_j}},t_{s_{e_i}})\leq\infty$	$d(t_{e_{e_j}},t_{s_{e_i}})=a$	$a\leq d(t_{e_{e_j}},t_{s_{e_i}})\leq b$	-
during	$0<d(t_{s_{e_j}},t_{s_{e_i}})\leq\infty$	$d(t_{s_{e_j}},t_{s_{e_i}})=a$	$d(t_{s_{e_j}},t_{s_{e_i}})=a$	$a1\leq d(t_{s_{e_j}},t_{s_{e_i}})\leq a2$
	$0<d(t_{e_{e_i}},t_{s_{e_j}})\leq\infty$	$d(t_{e_{e_i}},t_{s_{e_j}})=a$	$d(t_{e_{e_i}},t_{s_{e_j}})=b$	$b1\leq d(t_{e_{e_i}},t_{s_{e_j}})\leq b2$
contains	$0<d(t_{s_{e_i}},t_{s_{e_j}})\leq\infty$	$d(t_{s_{e_i}},t_{s_{e_j}})=a$	$d(t_{s_{e_i}},t_{s_{e_j}})=a$	$a1\leq d(t_{s_{e_i}},t_{s_{e_j}})\leq a2$
	$0<d(t_{e_{e_j}},t_{s_{e_i}})\leq\infty$	$d(t_{e_{e_j}},t_{s_{e_i}})=a$	$d(t_{e_{e_j}},t_{s_{e_i}})=b$	$b1\leq d(t_{e_{e_j}},t_{s_{e_i}})\leq b2$
overlaps	$0<d(t_{s_{e_i}},t_{s_{e_j}})\leq\infty$	$d(t_{s_{e_i}},t_{s_{e_j}})=a$	$d(t_{s_{e_i}},t_{s_{e_j}})=a$	$a1\leq d(t_{s_{e_i}},t_{s_{e_j}})\leq a2$
	$0<d(t_{e_{e_i}},t_{e_{e_j}})\leq\infty$	$d(t_{e_{e_i}},t_{e_{e_j}})=a$	$d(t_{e_{e_i}},t_{e_{e_j}})=b$	$b1\leq d(t_{e_{e_i}},t_{e_{e_j}})\leq b2$
overlapped-by	$d(t_{s_{e_i}},t_{s_{e_j}})=0$ $d(t_{e_{e_i}},t_{e_{e_j}})=0$	$d(t_{s_{e_i}},t_{s_{e_j}})=a$ $d(t_{e_{e_i}},t_{e_{e_j}})=a$	$d(t_{s_{e_i}},t_{s_{e_j}})=a$ $d(t_{e_{e_i}},t_{e_{e_j}})=b$	$a1\leq d(t_{s_{e_i}},t_{s_{e_j}})\leq a2$ $b1\leq d(t_{e_{e_i}},t_{e_{e_j}})\leq b2$
starts	$d(t_{s_{e_i}},t_{s_{e_j}})=0$	$d(t_{s_{e_i}},t_{s_{e_j}})=0$	$d(t_{s_{e_i}},t_{s_{e_j}})=0$	$d(t_{s_{e_i}},t_{s_{e_j}})\leq a1$
	$0<d(t_{e_{e_i}},t_{e_{e_j}})\leq\infty$	$d(t_{e_{e_i}},t_{e_{e_j}})=a$	$a\leq d(t_{e_{e_i}},t_{e_{e_j}})\leq b$	$b1\leq d(t_{e_{e_i}},t_{e_{e_j}})\leq b2$
started-by	$d(t_{s_{e_j}},t_{s_{e_i}})=0$	$d(t_{s_{e_i}},t_{s_{e_j}})=0$	$d(t_{s_{e_i}},t_{s_{e_j}})=0$	$d(t_{s_{e_j}},t_{s_{e_i}})\leq a1$
	$0<d(t_{e_{e_j}},t_{e_{e_i}})\leq\infty$	$d(t_{e_{e_j}},t_{e_{e_i}})=a$	$a\leq d(t_{e_{e_j}},t_{e_{e_i}})\leq b$	$b1\leq d(t_{e_{e_j}},t_{e_{e_i}})\leq b2$
finishes	$d(t_{e_{e_i}},t_{e_{e_j}})=0$	$d(t_{e_{e_i}},t_{e_{e_j}})=0$	$d(t_{e_{e_i}},t_{e_{e_j}})=0$	$a1\leq d(t_{s_{e_j}},t_{s_{e_i}})\leq a2$
	$0<d(t_{s_{e_j}},t_{s_{e_i}})\leq\infty$	$d(t_{s_{e_j}},t_{s_{e_i}})=a$	$a\leq d(t_{s_{e_j}},t_{s_{e_i}})\leq b$	$d(t_{e_{e_i}},t_{e_{e_j}})\leq b1$
finished-by	$d(t_{e_{e_j}},t_{e_{e_i}})=0$	$d(t_{e_{e_i}},t_{e_{e_j}})=0$	$d(t_{e_{e_i}},t_{e_{e_j}})=0$	$a1\leq d(t_{s_{e_i}},t_{s_{e_j}})\leq a2$
	$0<d(t_{s_{e_i}},t_{s_{e_j}})\leq\infty$	$d(t_{s_{e_i}},t_{s_{e_j}})=a$	$a\leq d(t_{s_{e_i}},t_{s_{e_j}})\leq b$	$d(t_{e_{e_i}},t_{e_{e_j}})\leq b1$

required to have identical start times as e. g. for the *starts* operator, they should arrive at the same time unit, thus their upper bound is zero. Events have to be kept for one time unit to ensure that all possible matches take place, then they can be discarded. If the start time $t_{s_{e_i}}$ of e_i is required to be less than that of e_j as e. g. for the *containing* operator, event e_i has to be kept, because matches with e_j can still take place. In this case it needs to be determined, if an event can be discarded considering its end time-point. The upper bounds $d^{ue}_{e_i,e_j}$ of an event e_i's end time-point used in an operator with event e_j, both events arriving at their start time-points, can be estimated as follows:

$$d^{ue}_{e_i,e_j}=\begin{cases}0 & \text{if } t_{e_{e_i}}=t_{e_{e_j}}\vee t_{e_{e_i}}=t_{s_{e_j}}\vee t_{e_{e_j}}=t_{s_{e_i}}\vee\\ & (t_{e_{e_i}}\leq t_{e_{e_j}}\wedge t_{s_{e_j}}\leq t_{e_{e_i}})\\ -\infty & \text{if } t_{e_{e_i}}>t_{e_{e_j}}\\ \infty & \text{otherwise}\end{cases} \tag{2}$$

Analogous to the calculation for start time-points, the upper bound of events is zero if they need to have equal end time-points as e. g. the *finishes* operator. However, events also have an upper bound of zero when their end time-points are required to be identical with the start time-points of their matching partners as e. g. for the *meets* operator. The same applies when the start time-points of events are required to be equal to the end time-points of matching partners as e. g. for the *met-by* operator. Moreover, the upper bound of an event's end time-point is also zero, if its end time-point is less than the matching event's end time-point, but it is greater than the matching event's start time-point as e. g. for the *during* operator. All these three calculations leading to an upper bound of zero can be explained by the fact that events are ordered by their start timestamp. Thus in all three cases all matching events should arrive at the same time unit as the event ends or have already arrived before the event's end time-point. An upper bound of $-\infty$ occurs if the end time-point of an event is greater than the end time-point of a matching event as e. g. for the *before* operator, i. e. all matching events arrived at some time in the past. Otherwise the upper bound is infinite.

For eight of Allen's operators, a restriction on the start time-point is given, the events can be instantly deleted after matching or be deleted after one time unit. The other five operators do not set any restrictions on an event's end time-point and hence result in an unlimited upper bound, since any of the events e_j

occurring after e_i could lead to a match. The lower bounds match the upper bounds.

For all operators with an unlimited upper bound on an event's start time-point, the bounds on end time-points of the event can be used. Only for the *before* operator no bounds can be determined using a combination of bounds on start and end time-points.

Events Arriving at their End Time-point. When events arrive at their end time-points, these time-points need to be considered for lifetime calculation. In this case, the upper bound $d^{ue}_{e_i,e_j}$ of an event e_i's end time-point with respect to an event e_j, both events arriving at their end time-points, can be estimated as follows:

$$d^{ue}_{e_i,e_j} = \begin{cases} 0 & \text{if } t_{e_{e_i}} = t_{e_{e_j}} \\ -\infty & \text{if } t_{e_{e_i}} > t_{e_{e_j}} \\ \infty & \text{otherwise} \end{cases} \quad (3)$$

Similar to the upper bounds of the start timestamps, if the end timestamp $t_{e_{e_i}}$ of event e_i is greater than the end timestamp $t_{e_{e_j}}$ of event e_j as e. g. for the *after* operator, the upper bound is $-\infty$, all matching events should have already arrived and the event could therefore be discarded. If the end time-points of two events are required to be equal, the upper bound is zero as e. g. for the *finishes* operator, events can be discarded after one time unit, since all matching events should arrive simultaneously. However, if none of these two conditions applies as e. g. for the *meets* operator, no other restriction can be made, since it is always possible that events with a higher end time-point arrive later which fulfil all start time-point constraints. As a result, there is a much higher number of operators for which no upper bound can be defined compared to events arriving at their start time-point. One way to deal with this problem is the definition of a maximum event lifetime.

2.4.2 Bounds on Parametrized Operators

Previously, we introduced bounds for Allen's original qualitative operators. Now, the bounds for our quantitative operators are stated in Table 2. The operators are given with their parameters, the bounds and their reasons. The lifetimes of events e_{n+1} or tuples $(e_1,...,e_n)$ can be calculated analogous to the method suggested by Teodosiu and Pollak. Again, a distinction has to be made for events arriving at their start time-point and those arriving at their end time-point.

In the first case, the lower and upper bounds for temporal distances $d^{ls}_{e_i,e_j}, d^{us}_{e_i,e_j}$ based on the operators an event is used in can be calculated for Allen's operators as described earlier or by taking the end bounds from Table 2 for the parameterized operators. These bounds are then used to calculate an event's upper bounds based on all its relationships. Whenever no bound is defined on an event's start time-point, the bounds on its end time-point can be used.

In the second case, lower and upper bounds for temporal distances $d^{le}_{e_i,e_j}, d^{ue}_{e_i,e_j}$ can be calculated for Allen's operators as described earlier or by taking the end bounds from Table 2 for parameterized operators. From these times, the lifetime can be determined for all events or tuples as suggested by Teodosiu.

The constraints an event has to fulfil depend on the complex event definitions it occurs in. The calculation of an event's lifetime is done based on these constraints. Therefore, the lifetime calculation can be done only once, when a rule defining a complex event is added to an RBS or updated in an RBS, e. g. during initialisation phase of an RBS, and then be stored to avoid recalculation. Thereby the lifetime of an event and the lifetime of a tuple arriving at a node can be stored locally at this node. Then, a garbage collection mechanism can access this time whenever needed in order to discard unneeded events from this node.

3 RELATED WORK

Next, we describe ideas of garbage collection and relative constraints in the Rete algorithm.

Berstel (Berstel, 2002) described the adaptations of the Rete algorithm used in ILOG JRules. He used time-point semantics and *before* and *after* operators are incorporated into Rete. The event concept is similar to ours, but distinguishes itself by using time-point instead of interval-based semantics.

TPS (Maloof and Kochut, 1993) is a traditional production system based on the Rete algorithm which is extended with temporal knowledge by storing past and developing events in a temporal database. An interval-based timestamp semantics is used. The system supports detection of events occurring *during*, *before* or *after* other events. However, operator semantics as well as conceptual details remain unclear.

HALO (Herzeel et al., 2007) is a logic-based pointcut language based on an extension of the Rete algorithm. A garbage collection mechanism deals with facts that are relevant only for a single time step or a certain time. Facts that are irrelevant from the start are not dealt with. The garbage collection is performed as part of a node's functionality, not by a separate mechanism. However, the paper does not explain in detail, how this garbage collection mechanism works or how it is determined if an element can be removed, just an example is given.

Table 2: Lower and upper bounds of an event based on its usage within a parameterized operator.

	Start bounds		End bounds		
Operator	$d^{ls}_{e_i,e_j}$	$d^{us}_{e_i,e_j}$	$d^{le}_{e_i,e_j}$	$d^{ue}_{e_i,e_j}$	**Reason**
e_i is equal to $e_j(a)$	$-a$	a	$-a$	a	$\lvert d(t_{s_{e_i}}, t_{s_{e_j}})\rvert \leq a$ $\lvert d(t_{e_{e_i}}, t_{e_{e_j}})\rvert \leq a$
e_i is equal to $e_j(a,b)$	$-a$	a	$-b$	b	$\lvert d(t_{s_{e_i}}, t_{s_{e_j}})\rvert \leq a$ $\lvert d(t_{e_{e_i}}, t_{e_{e_j}})\rvert \leq b$
e_i starts $e_j(a)$	0	0	$-a$	a	$d(t_{s_{e_i}}, t_{s_{e_j}}) = 0$ $\lvert d(t_{e_{e_i}}, t_{e_{e_j}})\rvert = a$
e_i starts $e_j(a,b)$	0	0	$-b$	b	$d(t_{s_{e_i}}, t_{s_{e_j}}) = 0$ $a \leq d(t_{e_{e_i}}, t_{e_{e_j}}) \leq b$
e_i starts $e_j(a_1,b_1,b_2)$	$-a_1$	a_1	$-b_2$	b_2	$d(t_{s_{e_i}}, t_{s_{e_j}}) \leq a_1$ $b_1 \leq d(t_{e_{e_i}}, t_{e_{e_j}}) \leq b_2$
e_i during $e_j(a)$	$-a$	a	$-a$	a	$d(t_{s_{e_i}}, t_{s_{e_j}}) = a$ $\lvert d(t_{e_{e_i}}, t_{e_{e_j}})\rvert = a$
e_i during $e_j(a,b)$	$-a$	a	$-b$	b	$d(t_{s_{e_i}}, t_{s_{e_j}}) = a$ $a \leq d(t_{e_{e_i}}, t_{e_{e_j}}) = b$
e_i during $e_j(a_1,a_2,b_1,b_2)$	$-a_2$	a_2	$-b_2$	b_2	$a_1 \leq d(t_{e_{e_i}}, t_{e_{e_j}}) \leq a_2$ $b_1 \leq d(t_{e_{e_i}}, t_{e_{e_j}}) \leq b_2$
e_i finishes $e_j(a)$	$-a$	a	0	0	$d(t_{s_{e_i}}, t_{s_{e_j}}) = a$ $\lvert d(t_{e_{e_i}}, t_{e_{e_j}})\rvert = 0$
e_i finishes $e_j(a,b)$	$-b$	b	0	0	$a \leq d(t_{s_{e_i}}, t_{s_{e_j}}) \leq b$ $d(t_{e_{e_i}}, t_{e_{e_j}}) = 0$
e_i finishes $e_j(a_1,b_1,b_2)$	$-b_2$	b_2	$-a_1$	a_1	$b_1 \leq d(t_{e_{e_i}}, t_{e_{e_j}}) \leq b_2$ $d(t_{e_{e_i}}, t_{e_{e_j}}) \leq a_1$
e_i overlaps $e_j(a)$	$-a$	a	$-a$	a	$d(t_{s_{e_i}}, t_{s_{e_j}}) = a$ $\lvert d(t_{e_{e_i}}, t_{e_{e_j}})\rvert = a$
e_i overlaps $e_j(a,b)$	$-a$	a	$-b$	b	$d(t_{s_{e_i}}, t_{s_{e_j}}) = a$ $d(t_{e_{e_i}}, t_{e_{e_j}}) = b$
e_i overlaps $e_j(a_1,b_1,b_2)$	$-a_2$	a_2	$-b_2$	b_2	$a_1 \leq d(t_{e_{e_i}}, t_{e_{e_j}}) \leq a_2$ $b_1 \leq d(t_{e_{e_i}}, t_{e_{e_j}}) \leq b_2$
e_i meets $e_j(a)$		$t_{e_{e_i}} - t_{s_{e_i}}$	$-a$	a	$\lvert d(t_{e_{e_i}}, t_{s_{e_j}})\rvert \leq a$
e_i before $e_j(a)$			$-a$	a	$d(t_{e_{e_i}}, t_{s_{e_j}}) = a$
e_i before $e_j(a,b)$			$-b$	b	$a \leq d(t_{e_{e_i}}, t_{s_{e_j}}) \leq b$

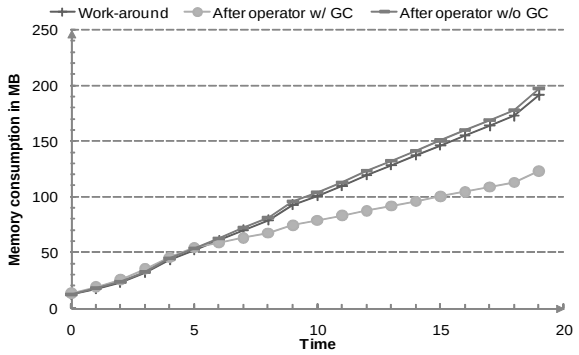
4 EVALUATION

We realized our proposed concepts for relative temporal constraints and garbage collection in Rete as an extension of the rule-based system JBoss Drools (Red Hat, Inc., 2007) version 4.0.3.

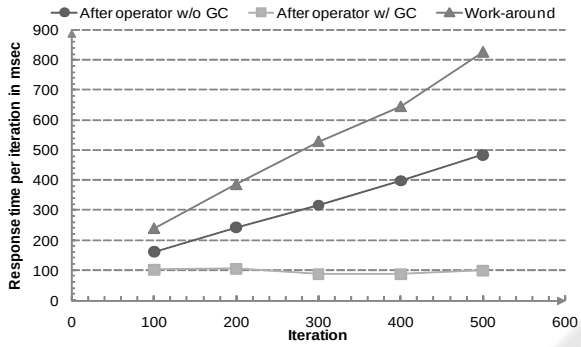
In our non-functional validation, we studied response times and memory usage of our system using our operators with and without garbage collection as well as a work-around without garbage collection using selections instead of our operators. We conducted all our experiments on a Dual Core AMD Opteron CPU (2.4 GHz) with 3.6 GB RAM running Linux 2.6.22. Here, we will show the results for the *after* operator exemplary.

We used two events of type *EventA* and *EventB* with attributes *x*, *start* and *end* denoting an event identifier and an event's start and end time-point. All events have a duration of two. All attributes have integer values. We used a rule that reacts to all events of type *EventB* that occur between 1 and 2 time units after an event of type *EventA*.

For the memory tests, we inserted an *EventA* and 20.000 *EventB* at each time unit observing 20 time-points (Figure 1 (a)). Then, we waited five seconds and advised the Java garbage collector explicitly to become active to ensure deterministic results. We notice that the memory requirements are lower when using the garbage collection compared to not using garbage collection.



(a) Memory consumption



(b) Response times per iteration

Figure 1: Comparison of memory consumption and response times per iteration for detection of the *after* operator; w/ GC – with garbage collection, w/o GC – without garbage collection.

We measured response time of our system analogous using the same rule as above. We advanced the time of our pseudo clock 5 times counting 100 iterations each time, inserting 1.000 events of each type in each iteration. Then, we divided the resulting time by the number of iterations performed at this measurement to determine the average response time (Figure1 (b)). We notice that using garbage collection leads to a constant response time whereas otherwise a continuously increasing response time can be observed, since the number of events increases over time.

5 CONCLUSIONS

We presented a method of determining if events can still contribute to future matches based on the satisfiability of their temporal constraints. Our proposed method leads to an exact calculation, improved memory consumption and response time. Nevertheless, improvements are still possible. For instance, node sharing could be studied which implies that a node condition occurs in several rules, thus having different temporal constraints.

REFERENCES

Allen, J. F. (1983). Maintaining Knowledge about Temporal Intervals. *Communications of the ACM Press*, 26:832–843.

Berstel, B. (2002). Extending the RETE Algorithm for Event Management. In *TIME '02*, page 49, Washington, DC, USA. IEEE Computer Society.

Forgy, C. (1982). Rete: A Fast Algorithm for the Many Patterns/Many Objects Match Problem. *Artificial Intelligence*, 19(1):17–37.

Herzeel, C., Gybels, K., and Costanza, P. (2007). Escaping with Future Variables in HALO. In Sokolsky, O. and Tasiran, S., editors, *RV '07*, volume 4839 of *LNCS*, pages 51–62. Springer.

Luckham, D. (2002). *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems*. Addison Wesley Professional.

Maloof, M. A. and Kochut, K. (1993). Modifying Rete to Reason Temporally. In *ICTAI*, pages 472–473.

Red Hat, Inc. (2007). JBoss Drools. Online. http://labs.jboss.com/drools/.

Sánchez, C., Slanina, M., Sipma, H. B., and Manna, Z. (2005). Expressive Completeness of an Event-Pattern Reactive Programming Language. In *FORTE*, pages 529–532.

Teodosiu, D. and Pollak, G. (1992). Discarding Unused Temporal Information in a Production System. In *CIKM '92*, pages 177–184.