TESLA: A Formally Defined Event Specification Language

Gianpaolo Cugola
Alessandro Margara

Proceedings of the Fourth ACM International Conference on Distributed Event-Based Systems - DEBS '10

Motivation

- Distributed applications often require large amount of information to be **timely** processed.
- The traditional data processing models does not suit the timeliness requirements.(DBMS)

Introduction(1/2)

- There are two models emerged: Data Stream Processing & Complex Event Processing.

- Data Streaming processing (DSP) is a model based on database.

- Complex Event Processing (CEP) is more of message-oriented.

Introduction(2/2)

- This paper claim that DSP is not suited to recognize patterns with temporal relationship.
- CEP often oversimplified, which is hard to express desirable patterns.
- TESLA , a complex event specification language they proposed, provides high expressiveness and flexibility, by offering filters (content 、 temporal) and operation(negation 、 aggregates …).

Why a new language : a representative example

- Consider a sensor network, which the sensors will notify position, temperature and smoke.

- Suppose we want to teach the system to notify user when fire occurs. The notion of fire can be defined in many ways.

- Using the below 4 rules to illustrate some features an event processing language should provide

Cont'd

Sequence of event

Select single notifications

Select set of related notifications or *parameterization*

i. temperature higher than 45 degrees and some smoke are detected in the same area within 3 minute. The fire notification has to embed the temperature actually measured.

Select timing relationship

ii. temperature higher than 45 degrees is detected and it did not rain in the last hour.

negation

iii. there is smoke and the average temperature in the last 3 minute is higher than 45 degrees.

aggregates

iv. at least 10 temperature readings with increasing values and some smoke are detected within 3 min. The fire notification has to embed the average temperature of the increasing sequence.

Iteration

Select , parameterization , sequence , negation, aggregates, iteration

Problem with existing language

- A representative DSP language, CQL, has a key aspect of forgetting the order. So it is hard to do sequencing operation.
- CEP, however , has many restriction like forcing to capture only adjacent event, which making it impossible to express some rules(like i & iii)
- Also, CEP faces the ***event selection problem*** and ***event consumption problem***.

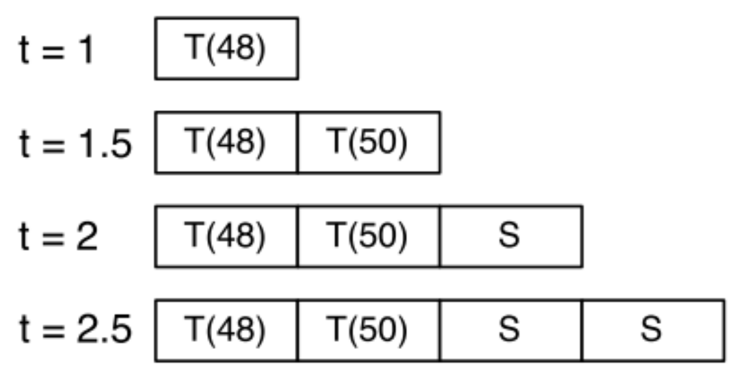
Cont'd

I. Consider rule (i), when t = 2, how many fire notification should be generated? {T(48) ,S} ? {T(50) ,S}? Both?

We call the problem of deciding how to combine events the ***event selection problem .***

II. Now what happens when t =2.5, where another smoke occur. Should the T notification is considered as "used", or they should be reconsidered again

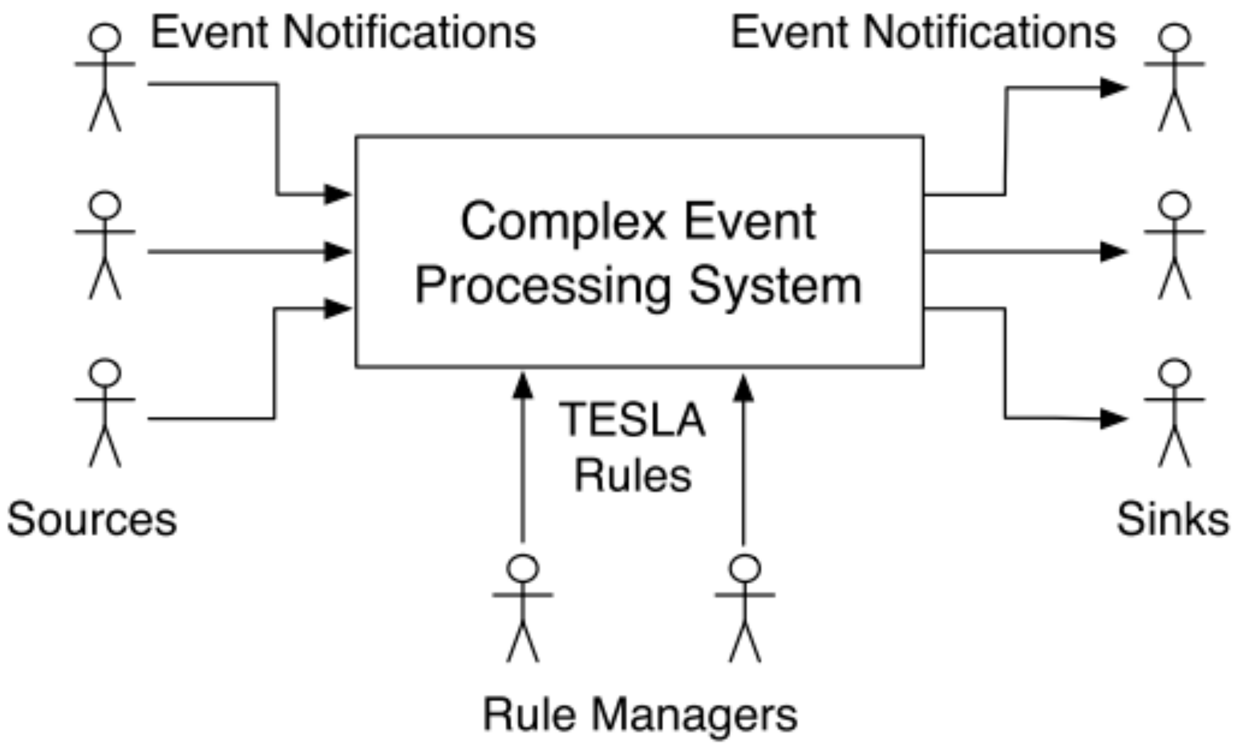
We call the problem of deciding invalid notification the ***event consumption problem***.



Overview - TRIO

- TESLA represents *Trio-based Event Specification Language*, where Trio is a first order, metric temporal logic.
- The special operator in Trio is temporal type operands and operator.
- Past(A, t) (resp. Futr(A, t)) , A holds t time units past (resp. future).
- $\mathrm{Alw}(A) = A \wedge \forall t(t > 0 \rightarrow \mathrm{Futr}(A, t)) \wedge \forall t(t > 0 \rightarrow \mathrm{Past}(A, t))$
  : Always A holds.
- $\mathrm{WithinP}(A, t1, t2) = \exists x(t1 \leq x \leq t1 + t2 \wedge \mathrm{Past}(A, x))$
  : Within the past t1 with length t2

TESLA event and rule model



Structure of the rules

$$\begin{array}{ll} define & CE(Att_1 : Type_1, ..., Att_n : Type_n) \\ from & Pattern \\ where & Att_1 = f_1, .., Att_n = f_n \\ consuming & e_1, .., e_n \end{array}$$

- Define a complex event(CE) and its structure.
- The pattern of simpler event leads to complex ones.
- Assign the attributes to CE which may depend on pattern.
- Last, decides which event should be invalidated.

Semantics of rules(1/3)

- First, introducing ***labels*** for events to differentiate them.
- Especially for complex events defined through TESLA rule (Assume events from source have unique labels).
- Claiming that a given of events can only satisfy a rule at most once(***uniqueness of selection theorem***).
- Leverage the claim by defining a ***lab*** function which returns new label taking two argument: rule ID & set of labels (labels that represent the event leading to this new event).
- For labels uniquely identify complex events, lab has to be injective.

Semantics of rules(2/3)

- Introducing Occurs(Type, Label).
- Two formulas:
  - If there are two notifications having same label, they must be the same type.
  - If an event with a label ‘ l ’ occurs, no other events with same label can occur at different time.
- Introducing attVal (Label, name)
- Extract value of a named attribute in a event represented by the label.

Semantics of rules(3/3)

- A generic TESLA rule trans to TRIO formula.

$$
\begin{aligned}
& Alw\ \forall l_1, .., l_m \in L, \forall n_1, .., n_n \in N \\
& ((Occurs(CE, lab(r, \{l_1, .., l_m\})) \leftrightarrow Pattern) \wedge \\
& (Pattern \rightarrow attVal(lab(r, \{l_1, .., l_m\}), n_1) = f_1) \wedge \\
& (Pattern \rightarrow attVal(lab(r, \{l_1, .., l_m\}), n_n) = f_n))
\end{aligned}
$$

- Every time when Pattern becomes true, CE occurs.
- Also, assigning value to CE's attributes.

Valid patterns(1/8)

- Event occurrence

$$\begin{array}{ll} define & CE(Att_1, .., Att_n) \\ from & SE(Att_x \; op \; Val_x) \\ where & Att_1 = f_1, .., Att_n = f_n \quad \triangleq \end{array}$$

$$(Occurs(CE, lab(r, \{l_1\})) \;\leftrightarrow$$
$$(Occurs(SE, l_1) \;\wedge attVal(l_1, Att_x) \; op \; Val_x)) \;\wedge$$
$$(Occurs(SE, l_1) \;\wedge attVal(l_1, Att_x) \; op \; Val_x) \;\rightarrow$$
$$(attVal(lab(r, \{l_1\}), Att_1) = f_1 \;\wedge .. \wedge$$
$$attVal(lab(r, \{l_n\}), Att_n) = f_n)$$

Time(Label): return time of the event having the label.

Valid patterns(2/8)

- Event composition (selection)
- each-within

$$define\ CE\ from\ A\ and\ each\ B\ within\ x\ from\ A \triangleq$$
$$Occurs(CE, lab(r, \{l_0, l_1\}) \leftrightarrow$$
$$(Occurs(A, l_0) \wedge WithinP(Occurs(B, l_1), Time(l_0), x))$$

- last-within

$$define\ CE\ from\ A\ and\ last\ B\ within\ x\ from\ A \triangleq$$
$$Occurs(CE, lab(r, \{l_0, l_1\}) \leftrightarrow$$
$$(Occurs(A, l_0) \wedge WithinP(Occurs(B, l_1), Time(l_0), x)$$
$$\wedge \neg\exists t \in (Time(l_1), Time(l_0)]\ Past(Occurs(B, l_2), t)$$
$$\wedge (\neg Past(Occurs(B, l_3), Time(l_1)) \wedge l_3 > l_1))$$

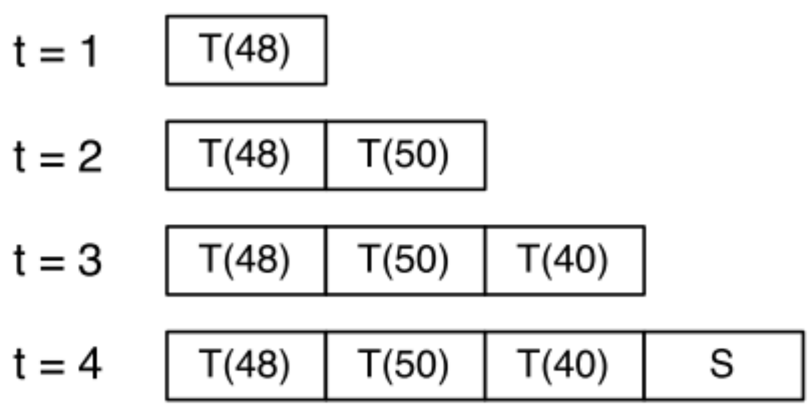
Assuming ordering !

- first-within

$$define\ CE\ from\ A\ and\ first\ B\ within\ x\ from\ A \triangleq$$
$$Occurs(CE, lab(r, \{l_0, l_1\}) \leftrightarrow$$
$$(Occurs(A, l_0) \wedge WithinP(Occurs(B, l_1), Time(l_0), x)$$
$$\wedge \neg\exists t \in [x, Time(l_1))\ Past(Occurs(B, l_2), t)$$
$$\wedge (\neg Past(Occurs(B, l_3), Time(l_1)) \wedge l_3 < l_1))$$

Valid patterns(3/8)

- Example:
- When t = 4, if each-within is used, T(48) and T(50) will combine with S. (multiple selection)
- If last-within is used, only T(50) will combine with S (single selection)



Valid patterns(4/8)

- Parameterization

$$
\begin{array}{ll}
define & Fire(Val) \\
from & Smoke(Area = \$x) \ and \\
 & each \ Temp(Val > 45 \ and \ Area = \$x) \\
 & within \ 5min \ from \ Smoke \\
where & Val = Temp.Val
\end{array}
$$

- Use Ș x to ensure that these events have same attribute value (Area attribute in this example).

Uniqueness of selection

- a set of events can be selected by a given rule only once.
- All TESLA rules joins the occurrence of a (complex) event to the occurrence of a pattern of (simpler) events, one of which must occur at the same time of the complex one, while the others occur in the past. This guarantees that a given rule r is satisfied by a set of events E only once, at time t.

Valid patterns(5/8)

- Timers: Timer(H = 9,M = 00,D = Friday)
- Negation: between 2 events or event with a duration.

$$
\begin{aligned}
&define\ D\ from\ A\ and\ each\ B\ within\ x\ from\ A\\
&and\ not\ C\ between\ B\ and\ A \quad \triangleq\\
&Occurs(D, lab(r, l_0, l_1)) \ \leftrightarrow\\
&(Occurs(A, l_0) \ \wedge \ WithinP(Occurs(B, l_1), x) \ \wedge\\
&\underline{\neg\exists t \in [Time(l_0), Time(l_1))} \ (Past(Occurs(C, l_2)), t))
\end{aligned}
$$

$$
\begin{aligned}
&C\ when\ A\ and\ not\ B\ within\ x\ from\ A \quad \triangleq\\
&Occurs(c, lab(r, l_0)) \ \leftrightarrow \ (Occurs(A, l_0) \ \wedge\\
&\underline{\neg\exists t \in [Time(l_0), Time(l_0) + x)} \ (Past(Occurs(B, l_1)), t))
\end{aligned}
$$

Valid patterns(6/8)

- Event consumption: consumption clause
- Introducing Consumed(rule ID, Label)
- Two formula:
  - Once an event has been consumed, it will keep consumed.
  - If an event is not captured, it will keep unconsumed.

Each rules has its own consumed list.

$$Alw\ \forall l \in L, \forall r \in \mathbb{N}$$
$$(Consumed(r,l) \rightarrow \forall\ t > 0, Futr(Consumed(r,l),t))$$

$$Alw\ \forall l \in L, \forall e, r \in \mathbb{N}, \forall S$$
$$((\neg \exists t > 0\ (Past(Occurs(e, lab(r,S),t))) \wedge\ l \in S)$$
$$\rightarrow\ \neg Consumed(r,l))$$

Valid patterns(7/8)

- Aggregates

$$Fun(X.Val)\ between\ A\ and\ B\ =\ Y\ \triangleq$$
$$\forall\ Set\ (\forall x(x \in Set\ \leftrightarrow \exists l \in L(x = < l, attVal(l, Val) >$$
$$\wedge\ withinP(Occurs(X, l), Time(B), Time(A))))$$
$$\rightarrow\ Fun(Set) = Y)$$

- Fun is the aggregates function
- Set includes all label-value couples of event X

Valid patterns(8/8)

- Iteration
- Example : suppose we want to capture every iteration of event A where the attribute never decrease, and notify another event B which contain number of A.

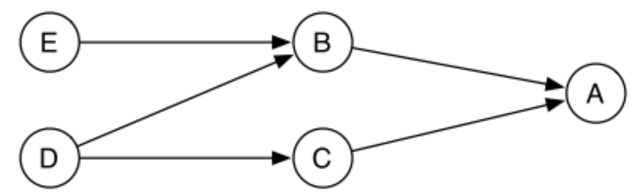
$$
\begin{array}{ll}
define & RepA(Times, Val) \\
from & A() \\
where & Times = 1 \ and \ Val = A.Val
\end{array}
$$

$$
\begin{array}{ll}
define & RepA(Times, Val) \\
from & A(\$x) \ and \ last \ RepA(Val \leq \$x) \ within \ 3min \\
 & from \ A \\
where & Times = RepA.Times + 1 \ and \ Val = \$x \\
consuming & RepA
\end{array}
$$

$$
\begin{array}{ll}
define & B(Times) \\
from & RepA() \\
where & Times = RepA.Times
\end{array}
$$

Event Detection Automata(1/5)

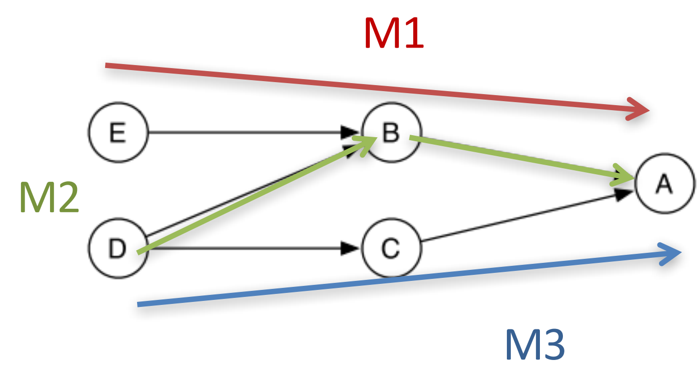
- Consider the Tesla rule below, which can transit into ordering graph.

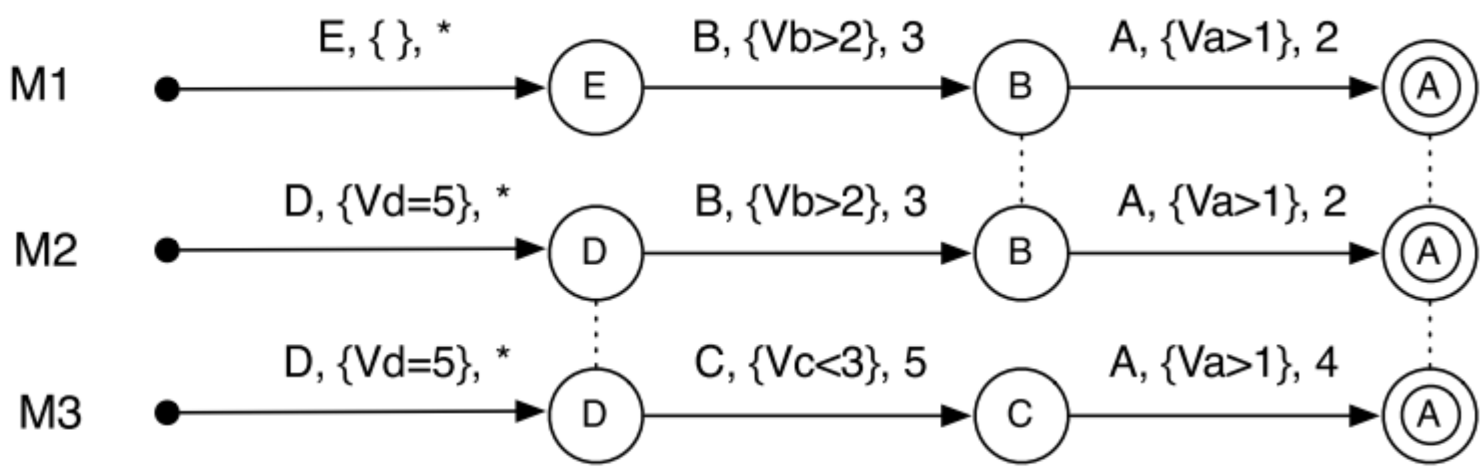


$$
\begin{array}{ll}
define & CE() \\
from & A(Va > 1) \\
 & and\ each\ B(Vb > 2)\ within\ 2\ min\ from\ A \\
 & and\ each\ C(Vc < 3)\ within\ 4\ min\ from\ A \\
 & and\ each\ D(Vd = 5)\ within\ 4\ min\ from\ B \\
 & and\ D\ within\ 5\ min\ from\ C \\
 & and\ each\ E()\ within\ 3\ min\ from\ B
\end{array}
$$

Event Detection Automata(2/5)

- Building automata model



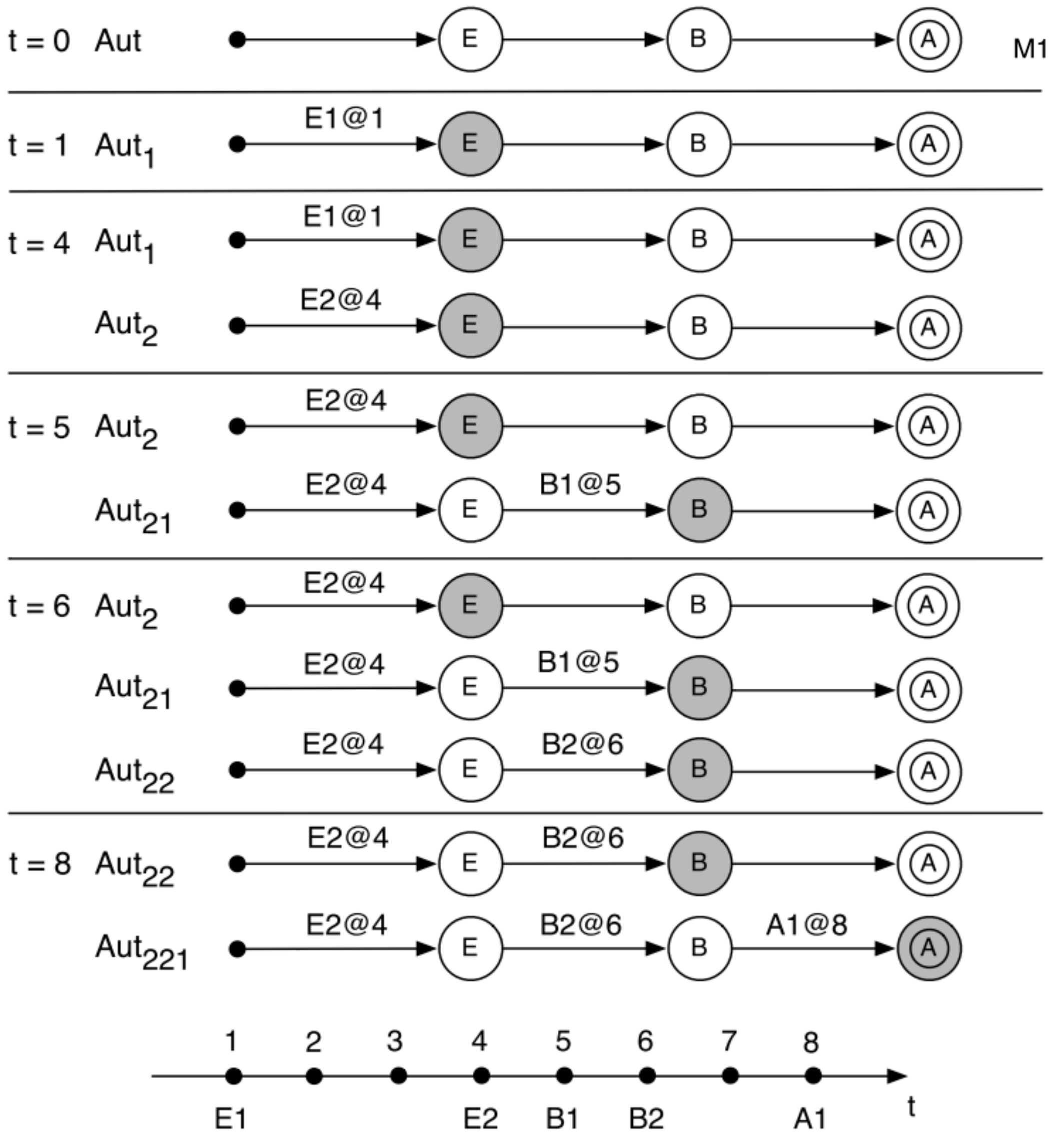


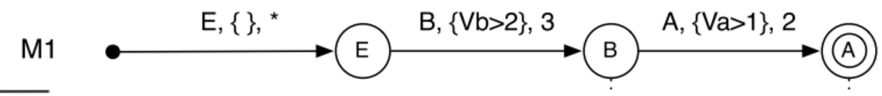
Event Detection Automata(3/5)

- Detecting simple sequences:

- Starts by creating single instance
- For each incoming event:
  - If it matches the current state, duplicate automata and enable transition to next state.
  - If it doesn’t match, ignore it.
  - If the maximum time of the state exceeded, delete it.

Event Detection Automata(4/5)





Event Detection Automata(5/5)

- Performance:
- Worst Case: # of automata grows exponentially.
- Average cases:
- Intel Core 2 2.53Ghz processor 98%, less than 700MB RAM, single threaded.
- 5000 rules with a total of 25000 automata states with a constant input rate 100 events / sec
- Peak: more than 1.5 million automata, 62000 events / sec input rate.

Conclusion

- TESLA provides a simple and compact syntax while offering high expressiveness and flexibility.
- fully customizable policies for event selection and consumption.
- allows TESLA to easily define event iterations without requiring an explicit Kleene operator.
- the first languages for CEP to offer a formal semantics, expressed using a temporal logic.
- introducing an event detection algorithm based on automata.