TESLA PARSER: A BRIEF INTRODUCTION

The general structure of a rule is as follows (this is Rule R1 from paper http://home.deib.polimi.it/cugola/Papers/trex.pdf), adapted so that it can be parsed from the Java client:

```
Assign 2000 => Smoke, 2001 => Temp, 2100 => Fire
Define Fire(area: string, measuredTemp: double)
From   Smoke(area=>$a) and
       each Temp([string]area=$a, value>45) within 300000 from Smoke
Where area:=Smoke.area, measuredTemp:=Temp.value
```

NAMING CONVENTIONS AND OPERATORS

- TESLA is case sensitive.
- Event names begin with a capital letter and follow the rules of identifiers in typical programming languages (i.e., letters, numbers and the “underscore” symbol are allowed).
- Attributes begin with a lowercase letter and follow the same rules.
- Parameters follow the rules of attributes but starts with the ‘$’ symbol.
- Names of aggregate functions are (AVG, MAX, MIN, SUM, COUNT) and are all capitals.
- A colon ‘ : ‘ separates the attribute names of the complex event from their types.
- Operator ‘ := ‘ is used to specify the value of complex event attributes int the “Where” clause.
- Operator ‘ => ‘ is used to introduce a parameter, providing its name.
- Selection policies are expressed through the ‘last’, ‘first’, and ‘each’ operators.
- Size of windows are expressed in milliseconds.
- Supported binary operators for computation are (‘+’, ‘-’, ‘*’, ‘/’, ‘&’, ‘|’)
- Supported binary operators for comparison are (‘>’, ‘<’, ‘>=’, ‘<=’, ‘=’, ‘!=’)
- Even if a predicate/negation/aggregate has no parameter/constraints, you must still write the opening/closing brackets after its name
-
- The rule starts with the special “Assign” clause, which associates event names with their corresponding, numeric value, used internally by the engine.
- The rule ends with a semicolon ‘ ; ‘

VALUES:

The parser automatically detects the type of a constant value, which can be int, float, bool or string.

Float values must always include a dot to separate integer from decimal part, such that value ‘10.0’ is assumed to be a float, while ‘10’ is an integer. Boolean values are ‘true’ and ‘false’, only. Finally, string values are expressed through quotation marks.

STATIC ATTRIBUTES/CONSTRAINTS

For a static assignment to be computed as static, you must express it with as a single constant. As an example, “where attr := 1 + 3 * 2” won’t be parsed as a static assignment, and the generated rule will be more complex to interpret. Instead “where attr := 7” wil be treated as static, and won’t require any complex computation by the engine.

PARAMETERS:

Selection expressions that include a parameter must always begin with the type of the expression. E.g., [string]area=$a.
References to attributes of different events can be expressed using a named parameter (e.g., ‘$a’) or referring to them explicitly suing EvtName.attrName (e.g., ‘Smoke.area’).

RENAMING EVENTS:

Events can be renamed using the ‘as’ keyword as in the following rule:

```
Define GrowingTemp(old: float, new: float)
From   Temp() as T1 and
       last Tamp(value > T1.value) as T2 within 60000 from T1
Where old := T1.value, new := T2.value;
```

NEGATIONS:

Negations can be expressed using the ‘not’ keyword as in the following examples:

```
Define Ce(area: string, measuredTemp: float)
From   Smoke(value => $a) and last Rain() within 500000 from Smoke and
       not Temp([int]value > $a * 2 + 17) between Smoke and Rain
Where area := "foo",
       measuredTemp := AVG(Rain.value([float]value > $a /2)) between Smoke and Temp;
```

```
Define Ce(area: string, measuredTemp: float)
From   Smoke(value => $a) and last Rain() within 500000 from Smoke and
       not Temp([int]value > $a * 2 + 17) within 100000 from Smoke
Where area := "foo",
       measuredTemp:=AVG(Rain.value([float]value > $a /2)) between Smoke and Temp;
```

AGGREGATES:

Aggregates can be expressed with the name of the corresponding function, followed by the EvtName.attrName couple that specifies the event atributes that must be used for the computation, then by the (optional) filtering predicates and finally by the selection policy. See the two examples of negation above.

CONSUMING:
Consuming can be expressed with the ‘Consuming ‘ keyword followed by a comma separated list of event names. As example:

```
Define Ce(area: string, measuredTemp: float)
From   Smoke(value => $a) and last Rain() within 500000 from Smoke and
       last Temp([int]value > $a * 2 + 17) within 500000 from Smoke
Where area := "foo",
       measuredtemp:=AVG(Rain.value([float]value > $a /2)) between Smoke and Temp
Consuming Rain, Temp;
```