Noname manuscript No.
(will be inserted by the editor)

Introducing Uncertainty in Complex Event Processing: Model, Implementation, and Validation

Gianpaolo Cugola · Alessandro Margara · Matteo Matteucci · Giordano Tamburrelli

Received: date / Accepted: date

Abstract Several application domains involve detecting complex situations and reacting to them. This asks for a *Complex Event Processing* (CEP) engine specifically designed to timely process low level *event notifications* to identify higher level *composite events* according to a set of user-defined *rules*. Several CEP engines and accompanying rule languages have been proposed. Their primary focus on performance often led to an oversimplified modeling of the external world where events happens, which is not suited to satisfy the demand of real-life applications. In particular, they are unable to consider, model, and propagate the *uncertainty* that exists in most scenarios. Moving from this premise, we present *CEP2U* (*Complex Event Processing under Uncertainty*),

Gianpaolo Cugola
Dipartimento di Elettronica, Informazione e Bioingegneria (DEIB)
Politecnico di Milano
Piazza L. da Vinci 32, Milan, Italy
E-mail: cugola@elet.polimi.it

Alessandro Margara
Dept. of Computer Science
Vrije Universiteit
De Boelelaan 1081A, 1081 HV Amsterdam, The Netherlands
E-mail: a.margara@vu.nl

Matteo Matteucci
Dipartimento di Elettronica, Informazione e Bioingegneria (DEIB)
Politecnico di Milano
Piazza L. da Vinci 32, Milan, Italy
E-mail: matteucci@elet.polimi.it

Giordano Tamburrelli
Faculty of Informatics.
Università della Svizzera Italiana
Via Buffi 13, Lugano, Switzerland
E-mail: giordano.tamburrelli@usi.ch

a novel model for dealing with uncertainty in CEP. We apply CEP2U to an existing CEP language –TESLA–, showing how it seamlessly integrate with modern rule languages by supporting all the operators they commonly offer. Moreover, we implement CEP2U on top of the T-Rex CEP engine and perform a detailed study of its performance, measuring a limited overhead that demonstrates its practical applicability. The discussion presented in this paper, together with the experiments we conducted, show how CEP2U provides a valuable combination of expressiveness, efficiency, and ease of use.

Keywords Complex Event Processing · Uncertainty Management · Uncertainty Modeling

1 Introduction

Several systems operate by observing a set of *primitive events* that happen in the external environment, interpreting and combining them to identify higher level *composite events*, and finally sending notifications about these events to the components in charge of reacting to them, thus determining the overall system's behavior. Examples are sensor networks for environmental monitoring [12, 22]; financial applications requiring a continuous analysis of stocks to detect trends [18]; fraud detection tools, which observe streams of credit card transactions to prevent frauds [41]; RFID-based inventory management systems, which perform a continuous analysis of registered data to track valid paths of shipments and to capture irregularities [46].

More in general, as observed in [29], the information system of every complex company can and should be organized around an *event-based core* that acts as a nervous system to guide and control the operation of other subsystems.

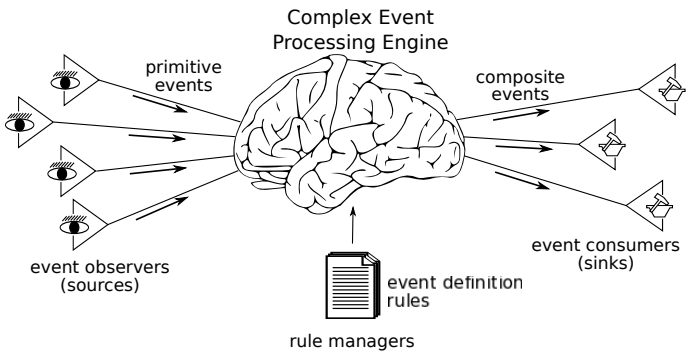


Fig. 1: The high level view of a CEP application

The general architecture of such an event-based application is shown in Fig. 1. At the peripheral of the system are the *sources* and the *sinks*. The

former observe primitive events and report about them, while the latter receive composite event notifications and react to them. The task of identifying so called composite events from primitive ones is referred as *Complex Event Processing (CEP)* and relies on a *Engine* which operates on the basis of a set of *rules* conceived and deployed by *rule managers*.

In the last few years several CEP engines and accompanying *rule languages* have been proposed, both from the academia and from the industry [17]. One of the main goals of such engines is to provide low delay processing of incoming primitive events. As we observed in [17], this often led to the design of oversimplified languages for rule definition, which are not well suited to capture the complexity of the aforementioned application scenarios. In particular, one of the main limitations of existing approaches is their inability to consider, model, and propagate the *uncertainty* that exists in most applications [7].

Starting from these premises, we developed CEP2U (Complex Event Processing under Uncertainty), a new model for dealing with uncertainty in complex event processing. CEP2U considers two types of uncertainty, namely uncertainty in the data coming from sources, and uncertainty in the induction step that derives composite events from primitive ones. The first form of uncertainty models the inherent imprecision of the information collected from sources. An example of this kind of uncertainty is the error introduced by a set of distributed sensors that measure temperature and humidity in a large area for weather forecast. CEP2U allows to represent such uncertainty and to propagate it into the generated composite events, e.g., the expected weather. The second form of uncertainty models the imprecision of rules, i.e., the possibility that rules do not completely reflect the behavior of the monitored environment. As an example, a rule that detects a fire event from smoke and high temperature may generate false positives if a smoker triggers a sensor for smoke detection.

In designing our model of uncertainty, we had four goals: (*i*) expressiveness: to make our model suitable for disparate real-world scenarios; (*ii*) generality: to allow easy integration of our model into existing rule languages; (*iii*) simplicity: to keep the model as simple and easy to use, read, and understand as possible; (*iv*) efficiency: to keep the overhead introduced by uncertainty management small w.r.t. event processing.

Organization of the paper. The rest of the paper is organized as follow. Section 2 presents background work in the field of complex event processing; it introduces the TESLA language and uses it to discuss the main features offered by existing systems. Section 3 presents the problem statement in details. The CEP2U model of uncertainty is described in Section 4, where we show, using TESLA as a practical example, how it can be easily integrated into the data and rule models of most existing CEP systems. Section 5 describes how we extended our T-Rex CEP engine [15] to fully support CEP2U. The performance of this prototype are thoroughly studied in Section 6 to show how the additional processing required by our uncertainty model introduces a lim-

ited overhead. Finally, Section 7 describes related work and Section 8 provides some conclusive remarks.

2 Background

2.1 Complex Event Processing

A heterogeneous world. Although complex event processing is a relatively new area of research, several CEP engines have been developed in the last few years, each one bringing its own data model, its own rule definition language, and its own processing algorithm and implementation.

Two main approaches have emerged and are currently growing in parallel [17]. On the one hand, there are systems developed by the database community, based on a processing model that is similar to that of relational databases, augmented with ad-hoc operators to support on-the-fly data processing. They are known as *Data Stream Management Systems (DSMSs)* [8]. Similar to SQL queries, the rules of DSMSs include operators that specify how to manipulate the (streaming) input information to transform it and generate one or more output streams.

On the other hand, the solutions proposed by the community working on event-based systems adopt rules that specify how composite events are defined starting from *patterns* of primitive ones. These kinds of rules do not express the processing steps to be performed on input events *explicitly*; on the contrary, the computation is *implicitly* specified by the pattern. The TESLA language [14] used in the remainder of the paper belongs to this second class.

A small world, after all. Despite their differences, these two approaches pose their roots on a common ground. Indeed, at the core of existing systems is usually a reduced set of abstract operators: *selection*, to isolate the primitive events relevant for processing based on their content; *combination* of multiple events based on their mutual relations in terms of content and occurrence time; *negation*, to identify events that must *not* occur in order to satisfy the rule; *aggregation*, to build composite events by putting together the content of the primitive events they come from; *production* of new (composite) events.

Next section presents these abstract operators in details, through a concrete example developed in TESLA, which is also the language we use in the remainder of the paper as a practical case to show how CEP2U is applicable to all the aforementioned abstract operators.

2.2 A Concrete Example

Tunnel Ventilation System (TVS) are crucial to guarantee safety in tunnels. To this end, they are constantly monitored by several sensors to detect possible failures, such as obstructions. In usual setups sensors are evenly distributed

along the tunnel and they measure the temperature and the concentration of oxygen in air. Now, consider a CEP application designed to detect TVS malfunctioning. Such application has to recognize critical situations starting from the raw data measured by sensors. Depending on the environment, the application requirements, and the user preferences, the presence of a TVS malfunctioning can be specified in many different ways. Here we present three possible definitions of the TVS malfunctioning event, and we use them to illustrate the abstract operators provided by event processing languages.

D1. There is a TVS malfunctioning when there is an oxygen concentration lower than 18%, while a temperature higher than 30 degrees was detected in the same sector of the tunnel in the previous 5 min. The malfunctioning notification has to embed the sector of the tunnel, together with the oxygen concentration and temperature actually measured.
D2. There is a TVS malfunctioning when there is a temperature higher than 30 degrees in absence of a traffic jam in the same tunnel sector.
D3. There is a TVS malfunctioning when there is an oxygen concentration lower than 18% and the average temperature in the last 5 min., in the same tunnel sector, was higher than 30 degrees.

To be implemented in a CEP system, all these definitions require the ability to *select* relevant notifications according to the values they carry (e.g., those about temperature higher than 30 degrees). Definition D1 also contains a specific example of *combination*, by coupling heterogeneous events (i.e., temperature and oxygen concentration) according to their relations on time (i.e., detected in the previous 5 min.) and on content (e.g., those occurred in the same sector of the tunnel). Similarly, definition *D2* introduces *negation* by requiring an event (the traffic jam) not to occur in a given interval; while definition *D3* introduces *aggregation* by requiring a specific function (average) to be applied to a specified set of elements (temperature readings in the last 5 minutes) to calculate the value associated with the composite event. Finally, each definition also states which notifications (i.e., TVS malfunction) have to be *produced* when the condition is verified. By doing this, it also specifies the internal structure of such notifications (e.g., TVS malfunctioning notification has to embed the tunnel sector and the oxygen and temperature readings).

2.3 TESLA in a Nutshell

This section leverages the example above to introduce TESLA and to show how the abstract operators above map to a concrete CEP language. The reader interested in a complete description of TESLA, including a rigorous discussion of its semantics, may read [14].

2.3.1 TESLA event and rule model

TESLA assumes that each event notification has a *type*, which defines the number, order, names, and types of the *attributes* that build the notification.

Moreover, by assuming that events occur instantaneously at some points in time, TESLA notifications also include a timestamp, which represents the time of occurrence of the event they encode.

Referring to the TVS example above, the TESLA event model allows to capture the fact that the air temperature at time 10 in the tunnel sector at 16.2km is 24.5C, using the following notification:

```
Temp@10(km=16.2, value=24.5)
```

Where `Temp` represents the type of the notification, which includes two attributes: a value that identifies the tunnel sector in which the temperature was measured, and the actual reading (both represented as floats).

As a final remark, notice that in the following we assume events to enter the system in timestamp order. This is a typical assumption in CEP systems, which delegate the ordering of events to external protocols. Existing solutions include waiting for a (fixed) time for out-of-order events before start processing, or introducing flexible heartbeats which take into account both clock synchronization and event propagation delays [42].

TESLA rules define composite events from simpler ones. The latter can be observed directly by sources (i.e., they can be primitive events) or they can be composite events defined by other rules[1]. Each TESLA rule has the following general structure:

```
Rule R
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
```

The `define` and `where` clauses, taken together, represent the *production* operator we identified above. Indeed, the former introduces the new composite event and its structure, while the latter defines the actual values for the attributes `att_1, .., att_n` of the new event using a set of *aggregation* functions `f_1, .., f_n`, which may depend on the arguments defined in `Pattern`. This pattern, part of the `from` clause, sets the conditions that lead to the composite event, *selecting* the primitive events and *combining* them according to well precise relationships. Finally, the optional `consuming` clause defines the set of events that have to be invalidated for further firing of the same rule.

2.3.2 TESLA by Examples

To present the operators supported by TESLA in an easy and clear way, we use the three definitions of TVS malfunctioning presented above, showing how they can be encoded in TESLA. In particular, definition *D1* can be represented by the TESLA Rule R1 below:

```
Rule R1
define TVS_Malfun(km:double, temp:double, ox:double)
from   Oxygen(concentr=<18% and km=$a) and
```

[1] This mechanism allows the definition of "hierarchies of events".

```
       last Temp($a-10 < km < $a+10 and value>30)
       within 5 min. from Oxygen
where  km=Oxygen.km and temp=Temp.value and ox=Oxygen.concentr
```

First, Rule R1 *selects* `Oxygen` and `Temp` events based on their content (i.e., `concentr<18%` and `value>30`). Then, it *combines* the two on the basis of their content (i.e., the value of attribute `km` put in relation using *parameter* `$a`) and their time of occurrence (i.e., `within 5 min`). Finally, it *produces* TVS malfunctioning events (`TVS_Malfun`) setting its attributes as specified by the `where` clause.

Notice how Rule R1 uses the `last-within` operator to bind each `Oxygen` event with the last `Temp` event observed. Alternative behaviors could be obtained by using other operators provided by TESLA for combining events. For example, the `first-within` operator would consider only the first `Temp` event observed within the specified time window, while the `each-within` operator would consider all available `Temp` events, producing a different TVS malfunctioning for each of them. The complete set of combination operators provided by TESLA enables fully customizable *selection policies* [17], thus allowing the domain experts to specify which events have to be considered when multiple choices are available. For a formal description of their semantics see [14].

The second definition of a TVS malfunctioning introduced above (i.e., *D2*) can be used to show how time-based *negations* are expressed in TESLA:

```
Rule R2
define TVS_Malfun(km:double, temp:double)
from   Temp(km=$a and value>30) and
       not TrafficJam($a-10 < km < $a+10)
       within 5 min. from Temp
where  km=Temp.km and temp=Temp.value
```

Rule R3 shows an example of *aggregation* function (i.e., `Avg`) to express definition *D3*.

```
Rule R3
define   TVS_Malfun(km:double, temp:double, ox:double)
from     Oxygen(concentr=<18% and km=$a) and
         30 < $t = Avg(Temp($a-10 < km < $a+10).value
         within 5 min. from Oxygen)
where    km=Oxygen.km and temp=$t and ox=Oxygen.concentr
```

Notice that the computed value of an aggregate can be used in the `from` clause, as in this case, to constrain the triggering of the rule, but it may also be used in the `where` clause, to set the attributes of the produced event.

3 Problem Statement

CEP rules are conceived to model —and capture— some aspects of interests of the real world. As for any modeling approach, accuracy is crucial. At the same time, our ability to capture a phenomenon is often affected by some form of uncertainty. Ignoring it may lead to incomplete, inaccurate, or even incorrect

decisions concerning the phenomenon itself. To achieve an effective management of uncertainty for CEP, the following three aspects must be considered:

- *identification* of the sources of uncertainty;
- *modeling* of uncertainty;
- *propagation* of uncertainty across the system.

Identification. The first step requires a careful analysis of the phenomenon to model, the way it is captured, and in general the environment where CEP is deployed, in order to identify the potential sources of uncertainty. For example, the limited accuracy of sensors may introduce uncertainty in data observations. In addition, even in presence of precise data observations, the behavior of the phenomenon under observation may depend from aspects hard to capture or changing over time, which if not considered may lead to imprecise models. As an example, when trying to detect fire in a building, the presence of smoke can be the key indicator, but it may also be unrelated with fire (e.g., if someone is smoking near the sensor), while high temperature is another indicator but its exact definition may change during the year (e.g., during the summer there is a non-negligible probability of having high temperature readings even without a fire).

Modeling. Once the sources of uncertainty have been identified they must be incorporated into the CEP system. The modeling phase aims at providing a sound mathematical foundation to represent uncertainty, let the CEP engine be aware of it, and manipulate it consistently. For example, probability theory can be used to model measurement errors, allowing the CEP engine to process uncertain values and combine them with other, possibly certain, ones.

Propagation. An uncertainty-aware CEP engine should produce a result characterized by an appropriate degree of uncertainty, consistent with: (*i*) the identified sources of uncertainty and (*ii*) the models adopted to represent them.

As mentioned in the introduction, the design of a CEP system should meet three key requirements: (*i*) expressiveness of the rule definition language, (*ii*) simplicity of this language, which results in a reduced effort in writing rules and increase readability and maintainability; (*iii*) efficiency in processing rules, to better support those scenarios, like financial applications, where a lower latency in detecting composite events results in a tangible advantage over competitors.

The same requirements should also drive the process of integrating uncertainty into CEP. Concretely, in presence of uncertainty, rules should remain concise and easy to write and read, while the overhead resulting from the introduction of uncertainty should be relatively limited.

4 Model of Uncertainty

This section describes the CEP2U model in details. As underlined above, the first step to model uncertainty consists in the identification of its sources. CEP2U focuses on two possible sources of uncertainty:

- *uncertainty in events*, i.e., the uncertainty deriving from an incorrect observation of the phenomena under analysis. This means to admit that the notifications entering the CEP engine can be characterized by a certain degree of uncertainty.
- *uncertainty in rules*, i.e., the uncertainty deriving from incomplete or erroneous assumptions about the environment in which the system operates. This means to admit that the CEP engine has only a partial knowledge about the system under observation, and consequently the CEP rules cannot consider all the factors that may cause the composite events they are in charge of detecting.

CEP2U models uncertainty in events using the theory of probability, while it exploits Bayesian Networks (BNs) [25] to model uncertainty in rules. In particular, it extends the model of events to include probabilistic data into event notifications, while it automatically builds a BN for each TESLA rule deployed in the system. Domain experts are expected to extend such BNs to capture a-priori knowledge about those aspects of the environment that cannot be directly observed by sources.

4.1 Uncertainty in Events

In this section we focus on uncertainty that affects event notifications, showing how we model it and how we propagate it during processing.

4.1.1 Modeling Uncertainty in Events

For each event e, CEP2U considers two forms of uncertainty:

i. the uncertainty regarding the content of e (i.e., regarding the values of its attributes).
ii. the uncertainty regarding the occurrence of e;

The first form of uncertainty stems from the fact that most attributes that builds event notifications hold a measure of some physical entity or phenomenon. These measures are inevitably affected by some degree of uncertainty, which derive from the inaccuracy, imprecision, and noise that affect sensors. CEP2U models this kind of uncertainty by considering the value of each attribute $Attr_i$ as a sample from a random variable $X'_i = X_i + \epsilon_i$, where X_i is the real, unknown value that would be measured in absence of errors, while ϵ_i is the measurement error.

CEP2U assumes that the probability distribution function (pdf) of ϵ_i is known. It depends upon the noise at sources, including fabrication randomness of sensors, inaccuracies in the measurement technique, etc. This information can be provided by the sources themselves (e.g., a sensor knows an estimate of its error and attaches it to the event notifications it produces) or it can be provided by a domain expert and integrated into event notifications before processing. Notice that our model supports generic pdfs, including discrete functions.

The second form of uncertainty is modeled through an estimate of the probability of occurrence of e. Events whose occurrence is assumed to be certain have a probability of 1, while events whose occurrence is not certain have a lower probability. In CEP2U, we assume all primitive events to be certain[2].

The definitions above result in extending the event model presented in Section 2.3 as follows:

i. For each event notification attribute $Attr_i$, the information received by the CEP engine is the couple: $\langle X'_i, pdf_i \rangle$, where X'_i is the observed value of $Attr_i$ and pdf_i is the probability distribution function of ϵ_i.
ii. Each event notification is augmented with an explicit probability of occurrence;

As a concrete example, consider our TVS scenario. The air temperature in a given tunnel sector at a specific time could be captured by the following event notification:

```
Temp@10 %1 (km = <16.2, U(-1, 1)>, value = <24.5, N(0,1)>)
```

Where %1 represents the probability of occurrence of the event `Temp` (it is certain that the measure was taken and consequently it is certain that the event occurred), while both the `km` and the `value` attributes have an associated uncertainty modeled through the pdfs of their measurement errors, which is a uniform (U) between -1 and 1 for `km` and a Gaussian (N) with mean 0 and variance 1 for `value`.

Before ending this section we need to discuss the aspect of events that was neglected so far: the time of occurrence. As the careful reader may have already noticed, we are assuming that the time at which events occur, and consequently the timestamp associated with event notifications (see Section 2.3.1), is a definite value. This is a debatable assumption. In principle, we could easily associate a degree of uncertainty to timestamps, modeling them as random variables instead of crisp values. On the other hand, we cannot ignore the fact that time represents a very critical aspect for every CEP system. Indeed, some of the fundamental CEP operators rely on timing constraints. This is the case of windows [8], sequence operators [17], and the `xxx-within` operator in TESLA, not to mention time-based aggregates.

[2] In particular, we assume that we receive all events (no false negatives) and that all received events actually occurred (no false positives). CEP2U can models the presence of false positives and negatives as part of the uncertainty in rules, as discussed in Section 4.2.2.

Associating uncertainty to a critical parameter as time could impact the semantics of rule languages, which often model cause-effect relationships and rely on temporal ordering to express causality. Furthermore, associating uncertainty to time would also negatively impact the efficiency of processing, with a potential explosion of the composite events captured, since primitives events "floating" in time may trigger much more rules than events whose occurrence time is precisely known. These considerations motivate our choice but we do not exclude to change it in the future.

4.1.2 Propagating Uncertainty in Events

Here we consider the abstract operators described in Section 2 and we show how uncertainty in primitive events propagates to composite events.

Selection. Consider the following, simplified version of Rule R1, which detects a malfunction every time a `Temp` event is detected that satisfies the two constraints: `km<17.1` and `value>30`:

```
define  TVS_Malfun()
from    Temp(km<17.1 and value>30)
```

When a `Temp` event enters the system whose attributes have an associated uncertainty, the satisfaction of the constraints in rule becomes uncertain, also. In such a situation we can compute the probability that each constraint is satisfied from the pdf of each attribute's measurement error ϵ_i.

As an example, assume that a `Temp` event is received at time 13 with the following measured values and associated errors:

```
Temp@13 %1 (km = <16.2, U(-1, 1)>, value = <31.8, N(0,1)>)
```

The probability that the first constraint in rule above is satisfied is: $P(X_{km} < 17.1)$ where X_{km} is the real, unknown value of attribute `km`. We know that:

$$X'_{km} = X_{km} + \epsilon_{km}$$

which means:

$$X_{km} = X'_{km} - \epsilon_{km} \sim U(16.2 - 1, 16.2 + 1) = U(15.2, 17.2)$$

Accordingly, the probability that X_{km} is lower than 17.1 corresponds to calculating $P(U(15.2, 17.2) < 17.1) = 0.9$. Similarly, for attribute `value` we have:

$$X_{value} = X'_{value} - \epsilon_{value} \sim N(31.8 - 0, 1) = N(31.8, 1)$$

Accordingly, the probability that it is greater than 30 is equivalent to calculating $P(N(31.8, 1) > 30) = 0.964$.

Finally, CEP2U assumes that the values of different attributes are independent from each other[3]. Under this assumption, the overall probability that the `Temp` event satisfies both constraints in the rule above is the product of the probability that each constraint is satisfied, i.e., $0.9 \cdot 0.964 = 0.868$. Since these are the only constraints in the rule, this is also the probability that the `TVS_Malfun` event is generated, i.e., the probability of occurrence that CEP2U associates to the composite event.

Combination. In several situations, composite events combine several primitive event by relating them on the basis of the values of their attributes.This case requires a different processing w.r.t. the selection case, as it relies on the comparison of multiple distributions. As an example, consider the following rule:

```
define   TVS_Malfun()
from     Oxygen(km=$a) and
         last Temp($a-10 < km < $a+10 and value>30)
         within 5 min from Oxygen
```

which requires the combination of `Oxygen` and `Temp` under the constraint that they occur in a neighborhood of 10km. As in the previous case, the evaluation of such constraint may be affected by uncertainty, since both event notifications may associate a measurement error to attribute `km`. To show how CEP2U addresses such case, assume we receive the following events:

```
Temp@10 %1(km=<10.5, N(0,1)>, value=<31.8, N(0,1)>)
Oxygen@12 %1(km=<10.3, N(0,2)>)
```

We already know from the previous example that the (selection) constraint on `value` is satisfied with probability 0.964. We now want to compute the probability that the constraint on `km` is satisfied. Assuming that the two readings of `km` are independent from each other[4], we want to compute the following probability:

$$P(-10 < X_{Temp.km} - X_{Smoke.km} < +10)$$

We know that:

$$\begin{aligned} X_{Temp.km} &= X'_{Temp.km} - \epsilon_{Temp.km} \\ &\sim N(10.5 - 0, 1) = N(10.5, 1) \\ X_{Oxygen.km} &= X'_{Oxygen.km} - \epsilon_{Oxygen.km} \\ &\sim N(10.2 - 0, 2) = N(10.2, 2) \end{aligned}$$

[3] In principle, we could remove this assumption and consider a multivariate pdf that involves all the attributes of incoming events, but this would complicate the model and it would require information that is usually unavailable at sources.

[4] This is a reasonable assumption, since the two readings come from different, independent sources. Only the occurrence of a TVS malfunctioning, whose probability is exactly what we are computing, may lead to correlated readings.

from which we derive:

$$X_{Temp.km} - X_{Oxygen.km} \sim N(10.5 - 10.2, 1 + 2)$$
$$= N(0.3, 3)$$

This probability can be computed as: $P(-10 < N(0.3, 3) < +10) = 0.999$. Combining this result with the previous one (about the selection operator) we calculate the overall probability of occurrence of `TVS_Malfun` as: $0.964 \cdot 0.999 = 0.963$.

Notice that the merging of random variables required to combine primitive events together, may result in complex pdfs that do not admit a closed formulation. CEP2U can deal with such cases exploiting Monte Carlo simulations, but they potentially require expensive computations to be performed online. However, their execution time may be tuned such that users can trade precision for efficiency [5]. In addition, we expect most applications to involve linear combinations of well known distributions, which can be analytically evaluated and computed (as in the example above).

Negation. Capturing the probability that a negation is satisfied is relatively easy. Indeed, assume we do not want an event of type `T` to occur in a given time window w. Also assume we receive n events of type `T` in w. Let us call p_i, $1 \leq i \leq n$, the probability that the i^{th} event occurs and satisfies the constraints stated by the rule (calculated as explained in the previous paragraphs). The probability that the negation constraint is satisfied equals to the probability P_{neg} that no `T` event occurs in w, which can be computed as follows:

$$P_{neg} = 1 - \prod_{i=1}^{n} p_i$$

As an example, consider the following rule, which is a simplified version of Rule R2 presented in Section 2:

```
define   TVS_Malfun()
from     Temp(km=$a and value>30) and
         not TrafficJam($a-10 < km < $a+10)
         within 5 min. from Temp
```

It includes a negation, requiring that no `TrafficJam` happens near the area where a high temperature was detected. Now assume we received the following three notifications:

[5] To better understand the overhead introduced by Monte Carlo simulations, we performed some experiments using curve fitting from randomly generated samples to approximate an unknown function. In presence of hundreds of samples, curve fitting required some hundreds of milliseconds to complete in our reference hardware. This is two order of magnitude higher than the typical processing time of a CEP engine (see Section 6 for more details). However, by reducing the granularity of the sampling intervals we could easily reduce the computation time to a few milliseconds. As future work, we plan to perform a detailed analysis of the tradeoffs between efficiency and precision.

```
TrafficJam@10 %1(km=<25.3, N(0,2)>)
TrafficJam@12 %1(km=<24.8, N(0,3)>)
Temp@14 %1(km=<10.5, N(0,1)>, value=<31.8, N(0,1)>)
```

By applying the process explained for combination operators, we obtain the following probability for the two `TrafficJam` events: 0.055 and 0.141. Using the formula above, the overall probability of occurrence of the `TVS_Malfun` event is:

$$P_{neg} = 1 - \prod_{i=1}^{2} p_i = 1 - (0.055 \cdot 0.141) = 0.992$$

The overall probability of `TVS_Malfun` is computed starting from the probability of `temp` and the probability of negation 0.992.

$$P_{\texttt{TVS\_Malfun}} = P_{neg} \cdot P_{\texttt{temp}} = 0.992 \cdot 0.964 = 0.956$$

Aggregation. Computing an aggregation requires applying a function f to the values extracted from a set of primitive events. As an example, consider the following simplified version of Rule R3:

```
define   TVS_Malfun()
from     Oxygen() and 30 < $t=Avg(Temp().value
         within 5 min. from Oxygen)
```

In this case the uncertainty of the aggregate value only depends on the uncertainty of the attribute `value` in `Temp`. In general, the theory of probability dictates how to compute the pdf of a random variable that results from applying a function `f` to a set $V = \{v_1, \ldots, v_n\}$ of random variables. In our example, such pdf represents the distribution of the average of all `value` in `Temp` events received within 5 minutes from `Oxygen`. For example, assume that n different `Temp` events have been received, with a value v_i and an error $\epsilon_i \sim N(0, \sigma_i^2)$. By remembering that $X_i' = X_i + \epsilon_i$, and that a linear combination of Gaussian pdfs is a Gaussian itself, the constraint on the average value of the attribute can be computed with the formula:

$$P(N(\sum_{i=1}^{n} \frac{v_i}{n}, \sum_{i=1}^{n} \frac{\sigma_i^2}{n^2}) > 30)$$

As already mentioned, in general things can be more complex, since combining random variables may lead to pdfs that do not admit a closed form. CEP2U may exploit Monte Carlo simulations to solve those cases.

Finally we note that, in the general case, a rule may also introduce additional (selection and composition) constraints that apply to the primitive events included in the computation of an aggregate as shown in the following rule.

```
define   TVS_Malfun()
from     Oxygen() and
         30 < $t = Avg(Temp($a-10 < km < $a+10).value
         within 5 min. from Oxygen)
```

In this case, the set of events to be considered for the computation is itself uncertain, as it depends on the satisfaction of additional constraints (i.e., the condition on `km`), which affects the distribution of the aggregated value. CEP2U supports this general case by considering the contribution of every possible subset S of primitive events (`temp` events in our example), weighted by the probability that all (and only) the events in S satisfy the selection and composition constraints. As a simple example assuming we receive two `temp` notifications t_1, t_2, in this case we compute the aggregate value as follows:

$$avg(t_1, t_2) \cdot P(t_1, t_2) + avg(t_1) \cdot P(t_1) + avg(t_2) \cdot P(t_2)$$

where $P(t_1), P(t_2)$ represent the probability that t_1 (respectively t_2) occurs, while $P(t_1, t_2)$ represents the probability that both t_1 and t_2 occur. Being t_1 and t_2 two random variables, their average is still represented by a random variable, thus the above computation generates another random variable[6].

Production. In all the examples above, we considered simplified versions of the rules presented in Section 2, in which the TVS malfunctioning event does not contain any attribute. Within this assumption, the uncertainty related to the measurement error only reflects on the probability associated to the occurrence of the composite events. In the general case, composite events include one or more attribute, whose value is determined starting from the attributes of primitive events. As an example consider again a simplified version of Rule R1:

```
Rule R1
define   TVS_Malfun(km:double, temp:double)
from     Oxygen(km=$a) and last Temp($a-10 < km < $a+10 and value>30)
         within 5 min. from Oxygen
where    km=Oxygen.km and temp=Temp.value
```

where the TVS malfunctioning event includes two attributes, `km` and `temp`. The first one is initialized to the tunnel sector of `Oxygen` and the second one to the value of `Temp`. CEP2U deals with such cases by propagating the uncertainty associated to the attributes of primitive events to the attributes of composite events. For instance, assume that we receive the following events:

```
Temp@10 %1(km=<10.5, N(0,1)>, value=<31.8, N(0,1)>)
Oxygen@12 %1(km=<10.3, N(0,2)>)
```

We already seen this situation when discussing composition and we computed the probability of `TVS_Malfun` as $P_{TVS\_Malfun} = 0.963$. Now, we also have to compute the attributes of the composite event with their pdfs. The resulting `TVS_Malfun` event becomes:

[6] Notice that, in the general case, the cost for computing the value of an aggregate can grow exponentially with the number of event notifications received. To limit the impact of this problem, it is possible to trade precision for efficiency, e.g., by approximating to 0 the occurrence probability of an event when it is sufficiently low (below a certain threshold) and to 1 when it is sufficiently high (above a certain threshold). The thresholds can be chosen based on the requirements in terms of precision and processing time.

```
TVS_Malfun@12 %0.963(km=<10.5, N(0,2)>, temp=<31.8, N(0,1)>)
```

In most cases, we expect that the values of the attributes in composite events are copies of primitive event attributes (as in the example above) or linear combinations of them. Not only CEP2U supports these simple operations, but also extends to the general case of non linear operations over different pdfs by relying on approximate distributions as discussed for combinations and aggregations.

Hierarchies of events. Several existing languages for CEP, including TESLA, allow the definition of rules that consider some composite events as their input to produce other (higher level) composite events. This enables rules manager to create *hierarchies of events*.

Consider for instance our tunnel ventilation system scenario. A user may want to define a rule that combines several `TVS_Malfun` events to detect a `Danger` (higher level) composite event.

Managing uncertainty in presence of hierarchies of events may become complex. Let us consider a rule R1, which generates a composite event `CE` used by another rule R2. In principle, the evaluation of uncertainty carried on in rule R2 over `CE` could depend from the computation performed in R1 to generate `CE`. An example of this situation is reported below:

```
Rule R1
define   CE(attr:double)
from     SE(attr>10)
where    CE.attr = SE.attr

Rule R2
define   CE2()
from     CE(attr>20)
```

Clearly, the evaluation of the selection constraint in rule R2 (i.e., `attr>20`) is not indepenent from the evaluation of the selection constraint in rule R1 (`attr>10`). Indeed, the attribute `CE.attr` is directly computed from `SE.attr`.

Intuitively, tracing the dependencies from rule to rule would severely impact on the complexity of processing, especially in presence of deep hierarchies, with multiple rules involved. Because of this, in CEP2U we decided to take a different approach, introducing two simplifying assumptions: (*i*) if an event `CE` is generated from a rule R1 and used as input for another rule R2, then R2 considers `CE` as occurred with probability 1; (*ii*) when a rule evaluates an event `CE`, it assumes that every computation involving the attributes of `CE` is independent from the process that led to the generation of `CE`.

Intuitively, through these assumptions CEP2U "forgets the past", ignoring the processing that led to the generation of an event `CE` while evaluating it. In other words, CEP2U considers all the events relevant for processing, including composite events generated in recursive rules, as if they were produced by external, independent, sources.

Since recursive rules are often used to build layers of abstraction, the assumption above can be interpreted by saying that CEP2U users "trust" the

results of lower levels when writing rules that define events at a higher level. While this may introduce some approximation in the computation of probability, it simplifies the semantics of rules and their design. Moreover, this enables CEP2U to process all input events in the same way, regardless if they are primitive events or composite events generated from other rules.

To mitigate the burden associated to recursive evaluation of composite events when their probability of occurrence is extremely low, we introduce an explicit (optional) clause to each rule, `min probability`. Such clause represents the minimum probability for considering the produced events as occurred and can be set by domain experts based on the application scenario at hand. If such a threshold is not reached the composite event is discarded for further processing and not delivered to interested sinks. Notice that this mechanism has the additional consequence of smoothing the effect of assumption (i).

A note about consumption. Another key feature considered in most CEP language, including TESLA, is the *consumption* policy [17,13]. It specifies which (primitive) events that fired a certain rule R have to be consumed, i.e., excluded from further evaluations.

In general, it is easy to understand that the model of uncertainty we introduced so far is not impacted by the mechanism of event consumption. Similarly, CEP2U has no impact on event consumption. Considering the specific case of TESLA, a definition of the currently supported consumption policies can be found in [14]. They apply seamlessly in presence of uncertainty.

4.2 Uncertainty in Rules

In the previous section we focused on the uncertainty associated with event notifications, under the assumption that the rules that detect composite events from primitive ones were definite and "certain". Here we relax this assumption and explain how CEP2U models the uncertainty deriving from rules using Bayesian Networks (BNs).

We adopted BNs to model the uncertainty in rules because they constitute a natural way to represent the dependencies between concepts. This perfectly fits our goal of modeling the uncertainty in the causal relationship between primitive and composite events.

As we will better explain later, one of the main drawbacks of BNs is represented by the complexity of their definition. To overcome this issue, CEP2U offers an automatic generation of BNs from rules and enables domain experts to modify and enrich them to better fit the scenario under analysis.

4.2.1 Bayesian Networks

Before giving the details of our solution, we provide an intuitive description of BNs. A BN is a graphical statistical model used to represent the conditional dependencies between random variables. Each BN is a directed acyclic graph (DAG): each node represents a random variable, while an edge from node

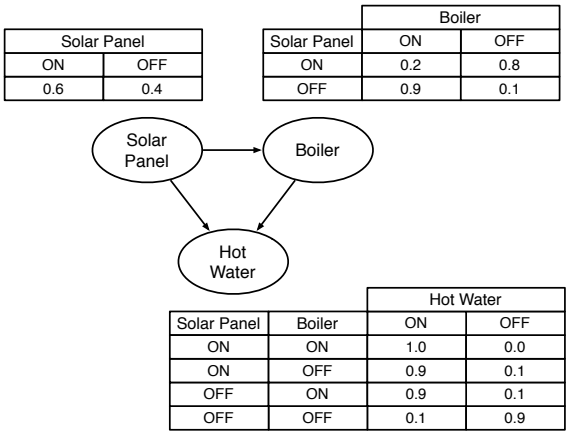


Fig. 2: Bayesian Networks: an example

N_1 to node N_2 represents a causal dependency between N_1 and N_2, i.e., N_1 causes N_2. Each node has an associated probability function that specifies the probability of each value that the node can assume as a function of the values assumed by parent nodes. BNs can be used to infer the expected values for one or more variables, given the values (or the a-priori distribution of values) of other variables in the network.

Fig. 2 shows a simple BN that models the availability of hot water using three random variables: `Solar Panel`, `Boiler`, and `Hot Water`.

Each of the three variables can assume two values, `ON` and `OFF`. The two edges entering `Hot Water` represent a causal dependency of this variable from both `Solar Panel` and `Boiler`. This dependency is quantified in the table associated to `Hot Water`: the availability of hot water is certain when both the boiler and the solar panel are `ON`, and highly probable ($P(\texttt{ON}) = 0.9$) when at one of them is `ON` and the other is `OFF`. When both the boiler and the solar panel are `OFF` then it is highly probable that there is no hot water (($P(\texttt{OFF}) = 0.9$). Notice that the network includes also a causal dependency between `Solar Panel` and `Boiler`, which models the presence of a controller to switch the boiler `ON` or `OFF` depending on the state of the solar panel. Finally, the state of the `Solar Panel` does not causally depend from any other random variable; accordingly, its table is filled with the a-priori probability that the solar panel is working correctly.

A BN can be used to answer queries about its variables and their causal relationships. For example, the network can be used to extract updated knowledge on the state of one or more variables when other variables are observed. In our case, we may observe that the solar panel is working (the value of `Solar Panel` is `ON` with probability 1) and use this information to compute the probability of having hot water equal to 98%.

4.2.2 Modeling Uncertainty in Rules through Bayesian Networks

As a concrete example to motivate the need of considering uncertainty not only in events but also in rules, consider Rule R1 again:

```
Rule R1
define   TVS_Malfun(km:double, temp:double, ox:double)
from     Oxygen(concentr=<18% and km=$a) and
         last Temp($a-10 < km < $a+10 and value>30)
         within 5 min. from Oxygen
where    km=Oxygen.km and temp=Temp.value and ox=Oxygen.concentr
```

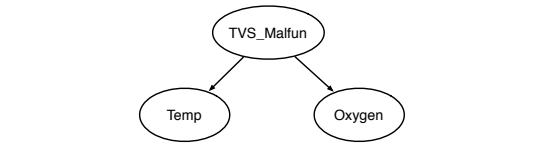
It describes how the presence of a `TVS_Malfun` event can be detected through the contemporary observation of high temperature and low oxygen concentration.

As it often happens in computer systems, this rule implicitly assumes a closed world in which only the three entities of oxygen, temp, and malfunctions exist, and there are no additional factors that influence their occurrence and their causal dependencies. However, reality is much more complex than this and a lot of other factors may actually influence the behavior of our tunnel. Some of these factors could be impossible to observe and measure, or it could be too complex to precisely monitor them. In general every modeling activity abstracts away those details that are considered marginally relevant, but in doing so it trades simplicity for precision, exposing to the risk that these omissions could result in incomplete models, possibly leading to wrong deductions. The ability to model the level of uncertainty present in rules is precisely what we offer to avoid this risk.

In particular, given a TESLA rule R: (i) CEP2U automatically *translates* R into a corresponding BN; (ii) rule managers or other domain experts are offered the chance to *enrich* such BN to include additional factors that may influence the occurrence of events; (iii) the updated BN is then *evaluated* to compute the probability of occurrence of composite events, taking into account the factors added in the previous step; (iv) the computed value is integrated with the results obtained by considering the uncertainty in events and *propagated* to the composite events generated by R.

Notice that steps (i–iii) occur at rule design time. Step (i) is performed by the CEP system every time a new rule is deployed. Rule managers may refine rules by enrichment —step (ii)—, automatically triggering step (iii). Step (iv) is the only one that occurs at run-time, while the CEP engine processes incoming events.

Translation. Each TESLA rule defines how the occurrence of a composite event can be detected from the observation of one or more primitive events. Put in other terms, it models a causal dependency among the composite events and the primitive ones. Moving from this consideration, CEP2U automatically translates each Rule R into a BN, which includes one node for each event (primitive or composite) in R and one edge (a causal relationship) connecting the composite event to each primitive event. As an example, Fig. 3 shows the BN obtained from Rule R1.



	Temp	
TVS_Malfun	YES	NO
YES	1.0	0.0
NO	0.0	1.0

	Oxygen	
TVS_Malfun	YES	NO
YES	1.0	0.0
NO	0.0	1.0

Fig. 3: The Bayesian Network Generated from Rule R1

Notice that this approach considers the composite event as the cause that determined the primitive events. This is reasonable in several situations, including our example of TVS malfunctioning, but there are scenarios in which what we observe (the primitive events) are the causes of the phenomenon that we are interested to detect (the composite event). For instance, this is the case of an application that measures seismic waves in a certain shore-side area to detect possible sea-quakes. In such situations, the rule managers may decide to revert the direction of the BN edges automatically created by CEP2U.

Moreover, the automatic translation assumes the composite event to be the *only* cause of *all* the primitive events. In other words, the presence of a composite event (`TVS_Malfun` in our example) always determines the occurrence of all primitive events (`Temp` and `Oxygen` in our example) with probability 1. Domain experts are allowed to modify this model by editing the BN, as described in the following (see the enrichment phase below).

As a final consideration, we want to recall that CEP2U uses BNs to model the contribution of uncertainty deriving from rules, exclusively. As a consequence, it abstracts away most aspects present in rules. In particular, each node in the BN obtained from a Rule R models a purely Boolean variable that represents, in abstract, the occurrence of an event that fully satisfies the constraints in R. For instance, node `Temp` in Fig. 3 represents the occurrence of a `Temp` event that satisfies the constraints `($a-10 < km < $a+10 and value>30)`.

Enrichment. During the enrichment phase, we allow rule managers or other domain experts to edit the BN, changing the direction of edges (see discussion above) if necessary, and introducing new nodes together with their causal dependencies with existing primitive and composite events.

Fig. 4 shows a possible enrichment of the BN obtained from Rule R1. In this case, we identified `TrafficJam` as an external cause of high temperature and low oxygen concentration. Accordingly, we added a new node representing such factor and connected it to both `Temp` and `Oxygen`. CEP2U admits generic (discrete and continuous) variables in BN, not only binary ones. In this case, we assumed `TrafficJam` may get three values: `HIGH`, `MEDIUM`, and `LOW`, and we added to the BN the a-priori distribution of these values. Notice that these distributions can be modified over time, for example to better represent variable scenarios such as seasonal trends.

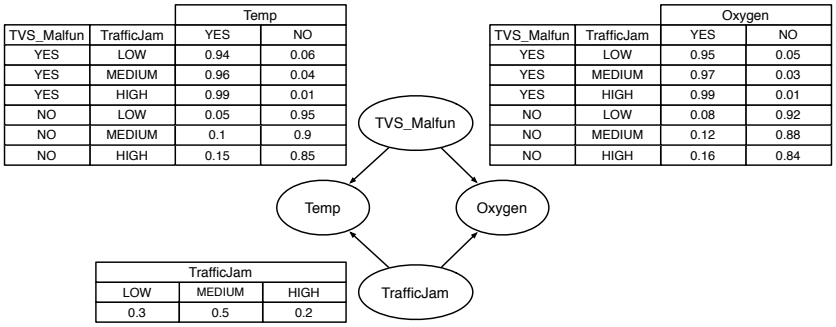


Fig. 4: Bayesian Network for Rule R1: an Example of Enrichment

Furthermore, when including additional nodes in the BN, the domain experts should also model how they influence existing ones. In Fig. 4, we modified the probability tables of `Temp` and `Oxygen` to take into account the possible presence of traffic jams.

Evaluation. The goal of the evaluation step is to determine the probability of occurrence of the composite event under the hypothesis that the primitive events have been observed. In practice, every node that represents a primitive event is set as happened (i.e., $P(\texttt{YES}) = 1$), and the BN is solved [26].

If the BN was a direct and "non-enriched" translation of a TESLA rule, the result of this step is to associate a probability of 1 to the occurrence of the composite event. For example, in the BN in Fig. 3 when both `Temp` and `Oxygen` are observed, the probability of occurrence of the `TVS_Malfun` event evaluates to one as well. This is not the case if the BN was enriched. Changing the probability tables of the various nodes or introducing new factors, like the `TrafficJam` in Fig. 4, changes the result of the evaluation step. As an example, solving the BN in Fig. 4 we get a probability of 98.70 for the occurrence of `TVS_Malfun`. Indeed, now the occurrence of `Oxygen` and `Temp` events cannot be directly and uniquely related to the occurrence of the `TVS_Malfun` event, as the former ones may also be caused by the presence of a traffic jam.

Propagation. Summing up the process explained so far, CEP2U produces two probabilities of occurrence for composite events. One by looking only at the uncertainty intrinsically related with events (see Section 4.1) and one by looking only at the uncertainty coming from rules, as just discussed. At this point, we need to merge them in a single probability value, but this is straightforward. Indeed, these two probabilities are independent by construction, accordingly we may combine them by simply multiplying the two. The result is the uncertainty we associate to the composite event (hence the name "propagation" of this step).

4.3 Discussing CEP2U Design

As mentioned in Section 3, when it comes to design an uncertainty model for CEP, there are some key requirements that must be carefully considered. Focusing on the rule language, they include a balanced trade-off between simplicity and expressiveness.

CEP2U allows engineers and domain experts to easily capture the two sources of uncertainty we identified in our analysis: the uncertainty coming from events and the uncertainty in rules.

In addition, CEP2U meets the separation of concern principle by considering them independently, as the uncertainty coming from rules does not affect the pdfs of the attributes in composite events, which are entirely and uniquely determined by the processing of the uncertainty coming with events. More specifically, the BNs we create are completely unaware of attribute values and only reason about event occurrences, while the processing of uncertainty associated with attributes assumes rules are definite and correct. This modeling choice allows a precise and clear separation of the factors that influence uncertainty in CEP, which keeps the model simple and easy to understand, while enabling its efficient evaluation at run-time, as discussed later on in Section 6.3.

Moreover, this approach maximizes the generality of our model, which has virtually no impact on the rule language. In integrating CEP2U with TESLA we only added a new (optional) `min probability` clause, the rest of the language remained the same.

Finally, the easiness of integrating CEP2U in TESLA, a language that includes all the typical CEP operators, hints at its general applicability to other CEP languages.

5 Implementation

The validation of CEP2U has been carried on by implementing the model and its TESLA incarnation into our T-Rex engine [15]. In this section we illustrate such implementation. The discussion will proceed by difference with respect to the previous implementation of T-Rex, focusing on the components introduced or modified to support our model of uncertainty.

The interested reader can find a detailed description of the architecture of T-Rex in [15]. Concerning specifically the event processing algorithm, i.e., the algorithm used to perform pattern matching over incoming events, we started from the CDP algorithm described in [16,30].

Fig. 5 shows the overall architecture of T-Rex, highlighting the components that have been added or modified to support uncertainty. In particular:

- we modified the processing algorithm of T-Rex to fully support the new CEP2U data model, which includes uncertainty on the occurrence and on the content of events;

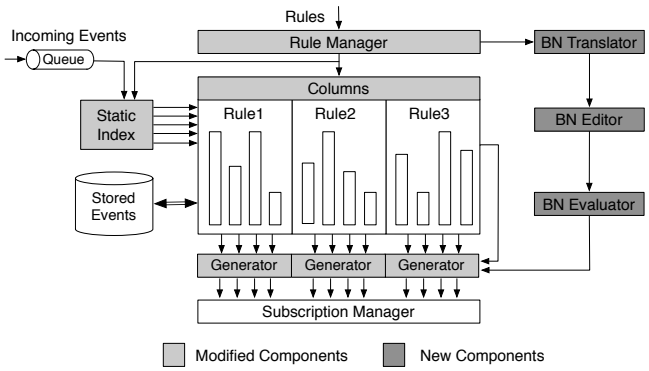


Fig. 5: Architecture of the T-Rex engine: components introduced and modified to support uncertainty.

– we implemented a `BN Translator`, `BN Editor`, and `BN Evaluator`, which automatically generate BNs from TESLA rules, support their editing, and evaluate them, respectively.

The first change is required to deal with the uncertainty in events during processing, to take into account possible measurement errors, and to propagate them into the generated composite events. The second change introduces the components to define and evaluate the uncertainty in rules.

5.1 Implementing Uncertainty in Events

The CDP processing algorithm adopted by T-Rex organizes events into ad-hoc data structures called `Columns`. More precisely, it generates a separate column for each primitive event appearing in each rule R. As an example, in Fig. 5 we see the engine handling 3 rules, each one involving 4 primitive events (and thus defining 4 columns). For efficiency reasons, a single copy of each event is stored in the `Stored Events` repository, while only pointers are actually present in `Columns`.

When a new event e enters the engine, the `Static Index` determines the set of columns it is relevant for based on e's type and content. In other words the `Static Index` implements the *selection* of primitive events, as described in Section 2. When the primitive events have been collected, the `Columns` component is responsible for performing the *combination* of such events, for evaluating *negations*, and for executing all the computations —including *aggregation*— that is necessary to understand if the pattern expressed in a rule is satisfied. When this happens, the `Generators` (one for each rule) are responsible for *producing* the corresponding composite events. Finally, the composite events enter the `Subscription Manager`, which is responsible for delivering them to the interested (local or remote) sinks.

As discussed in Section 4, considering the uncertainty in events impacts all the abstract operators of a CEP engine: selection, combination, negation, aggregation, and production. Accordingly, we had to modify the `Static Index`, `Columns`, and `Generators` components.

The `Static Index` received the most substantial changes. The goal of this component is to match incoming events against the selection constraints expressed by rules. This is the typical task performed by publish-subscribe systems, and several efficient algorithms have been proposed for it [3]. In absence of uncertainty we reused one of this algorithms, which exploits complex indexed structures to reduce the processing effort. Unfortunately, such mechanisms cannot be applied in presence of uncertain data. Accordingly, we re-implemented the `Static Index` from scratch using a simpler but more flexible algorithm that evaluates constraints and their satisfaction probability sequentially against each incoming event. Despite this significant change, the overhead of uncertainty handling over the selection step is limited. We will investigate this aspect in details in Section 6.

Uncertainty has a minor impact on the structure of the `Columns` component. Indeed, we simply modified the functions used to evaluate the constraints appearing in a rule —e.g., constraints on the combination of events, on negation, or on the values of aggregates— so that they do not return whether a constraint is satisfied or not (as a Boolean value), but its satisfaction probability (as a double).

Finally, the `Generators` have been modified to include the probability of occurrence into the generated composite events, and to include uncertainty on the values of attributes.

All the statistical computing required to consider and propagate the uncertainty in event attributes have been implemented on top of the math and statistical toolkit of the boost libraries [10].

As a final remark, we note that we had also to modify the `Rule Manager` component, which is responsible for receiving new TESLA rules and for creating all the data structures required to process them. This component underwent two (minor) updates: on the one hand it has been extended to consider the additional information in rules that result from introducing uncertainty (e.g., the threshold of probability required for producing composite events); on the other hand, it has been modified to deliver newly deployed rules to the `BN Translator`.

5.2 Implementing Uncertainty in Rules

To implement the uncertainty in rules, we exploited the Netica API [34] for representing and evaluating BNs. More precisely, we used it to build three components: the `BN Translator`, which translates new TESLA rules into BNs; the `BN Editor`, which enables the modification of BNs; the `BN Evaluator`, which uses BNs to evaluate the probability of occurrence of composite events.

The `BN Evaluator` provides the result of its computation to the `Generator`, which uses it while generating the composite event notifications (integrating it with the results obtained from the processing of events and their uncertainty).

The `BN Editor` consists of a front-end, which enables graphical editing of BNs, directly connected with the `BN Evaluator`, which acts as a back-end, performing the evaluation step every time a BN is modified.

Notice that our components use a representation of BNs that is enriched to store the nature of each node, which can denote a primitive event, a composite event, or an additional factor provided by domain experts. As explained at the end of Section 4.2, this information is used by the `BN Evaluator` to calculate the the occurrence probability of composite events.

5.3 Discussing CEP2U Implementation

In Section 4 we claimed the simplicity of CEP2U at the language level. This section illustrated the effort required to implement CEP2U in an existing processing engine.

Despite several components had to be modified, the changes introduced were typically small and concentrated in some specific functions (e.g., in the function for evaluating the value of aggregates, or for testing negations), while the general flow of execution remained unchanged.

This is possible because our model of uncertainty is almost orthogonal with respect to the processing algorithm that evaluates CEP rules. Indeed, the introduction of uncertainty in events simply transforms Boolean satisfaction of constraints into a probability of satisfaction, while uncertainty in rules is mostly managed outside the normal flow of CEP processing, delegating to BNs (at rule design time) large part of the computation (excluding the straightforward step —called "propagation" in Section 4.2.2— which consists of a simple multiplication).

6 Evaluation

As discussed in Section 2, performance is a key requirement for CEP engines: in many scenarios it is of primary importance to provide low delay processing to detect and notify critical situations as promptly as possible. Consequently, a good model for uncertainty should have a limited impact on the performance of the CEP engine.

Because of this, a large part of this section is devoted to study the impact of CEP2U on performance, by considering our implementation in T-Rex and measuring the overhead on event processing when uncertainty is managed. During our evaluation, we separately consider the uncertainty related to events and the uncertainty related to rules. As far as performance is concerned, the former is more critical, since it requires additional computation at

run-time, during event processing. Conversely, most of the processing related with uncertainty in rules happens at design time, when the BN associated to each rule is evaluated, the only computation occurring at run-time being a multiplication (see Section 4.2.2).

Beside performance, our evaluation also aims at understanding the added value that a CEP user gets in receiving information about uncertainty. To this end, we measure the accuracy of CEP2U in correctly identifying the occurrence of composite events.

6.1 Uncertainty in Events

All the tests described below were performed on a 2.8 GHz AMD Phenom II PC, with 6 cores and 8 GB of DDR3 RAM, running 64 bit Linux. We use a local client to generate events at a constant rate and to collect results. This way, the interaction with the CEP engine is realized entirely through local method invocations, which eliminates the impact of the communication layer on the results we collected and allows to focus on the raw performance of the engine. T-Rex is configured to take advantage of all available CPU cores, i.e., 5 out of 6, being 1 core used by the local client.

During our tests, we measure the average time required by T-Rex for processing a single input event e. In particular, we measure the interval from the point in time when e starts to be actively processed (i.e., when it exits the queue of incoming events in Fig. 5) to the instant when all the composite events that result from e are ready to enter the Subscription Manager. Given a specific workload, this metric enables us to compute the maximum input rate that T-Rex can handle. For example, with an average processing time of 1ms, we could theoretically process 1000 events per second. However, since 1ms is only an average measure, specific events may take longer: in presence of a finite input queue, this means that the engine can start dropping events before this theoretical rate[7]. For this reason, all our tests also measure the 99th percentile of the processing time.

We execute each test 10 times, computing and plotting the 95% confidence interval of each measure we collect.

The lack of standard benchmarking is a well known problem in the domain of event processing. Despite some scenarios have been defines (e.g., Fast Flower Delivery citepetzion-book), they are not well suited for a general assessment of CEP engines. This is mainly due to the heterogeneity of the solutions proposed. This problem is exacerbated in the case of uncertainty.

Because of this, in this section we decided to follow the structure of Section 4, and we defined several workloads to isolate the contribution of uncertainty over the processing of each abstract operator, i.e., selection, combination, negation, and aggregation. This solution based on microbenchmarking

[7] A detailed analysis on the impact of the input queue on performance is outside the scope of this paper, and can be found in [15].

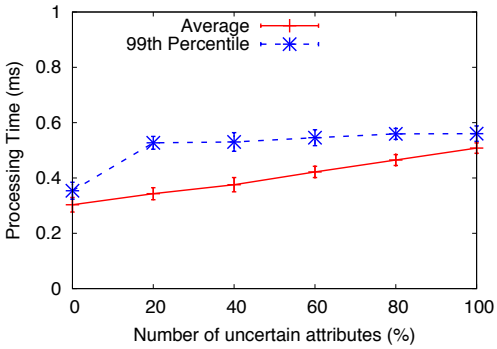


Fig. 6: Selection: processing delay

enables us to consider all the parameters that could impact on the performance and accuracy of CEP2U.

6.1.1 Selection

As we have seen, selection is the first, fundamental step performed by the CEP engine to isolate the primitive events relevant for each rule. To evaluate the performance of selection, we deployed 1000 rules, all having the following structure:

```
define  CompEv_i()
from    PrimEv_i(x-5 < value < x+5)
```

we considered 10 different types of primitive events, `PrimEv_1`, ..., `PrimEv_10`, each of them including a `value` attribute assuming values from 1 to 100. We deployed 100 different rules for each primitive event type, each one including a different constraint on `value` (i.e., a different value of `x`).

Afterward, we generated 10000 primitive events, selecting their type and value uniformly. We measured the performance of the engine while changing the percentage of primitive events whose `value` has an associated uncertainty, the remaining percentage of events being certain. The uncertainty of attributes (i.e., their measurement error ϵ) was modeled using Gaussian distributions.

Intuitively, this is a challenging workload, since it presents a (relatively) large number of rules and only a small number of event types. Indeed, T-Rex exploits the type of incoming events to efficiently distinguish between the rules that those events may potentially trigger and the rules that are not affected: a reduced number of event types negatively impacts the effectiveness of this approach, increasing the number of rules to consider and consequently the number of attribute-constraint comparisons to perform. Since the evaluation of a constraint against an attribute is influenced by the presence of uncertainty, the workload we adopt emphasizes impact of uncertainty on selection.

Fig. 6 shows the results we measured. First of all, we notice that T-Rex, even in presence of uncertainty, exhibits a low processing delay, below 0.6ms

with 1000 rules deployed on the engine. Second, we observe that the processing time increases linearly with the percentage of uncertain attributes but this growth is slow and the overall impact is limited.

As a final note, we observe how the presence of uncertain attributes quickly increases the 99^{th} percentile of the processing time, which remains almost constant when the percentage of events affected by uncertainty grows. This can be easily explained by noticing that only the events whose attributes have an associated uncertainty take longer to be processed.

6.1.2 Combination

As a second step, we evaluated the performance of our CEP2U implementation using the following rule, derived from our TVS scenario, which combines two primitive events:

```
define   TVS_Malfun_i()
from     LowOxygen_i(km=$a) and
         last Temp_i($a-10 < km < $a+10 and value>x)
         within 5 min. from LowOxygen_i
```

To stress the engine, each test deploys 1000 different rules with the same structure of the rule above but considering 10 different composite events (`TVS_Malfun_1`, defined from `Temp_1` and `LowOxygen_1`, ..., `TVS_Malfun_10`, defined from `Temp_10` and `LowOxygen_10`) and asking for a different minimum temperature (from 1 to 100). The value of the temperature of incoming `Temp_i` events is uniformly distributed between 1 and 100, while all events share the same `km` to maximize the probability of using events.

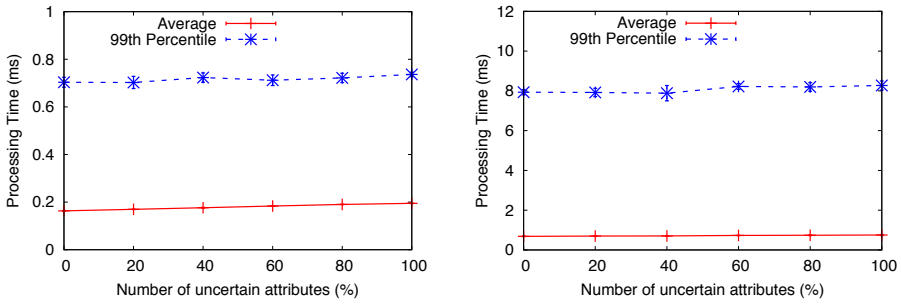
Given the impact of the selection policy on performance, we performed every test twice, once using the `last-within` operator (as shown in the rule above) and one using the `each-within` operator. In both cases, we tested three different workloads, generating respectively 10%, 50%, and 90% of `LowOxygen_i` events (the remaining events are `Temp_i`).

The presence of a higher number of event types w.r.t. the previous experiment (10 for `LowOxygen_i` and 10 `Temp_i`) explains the lower overhead induced by the presence of uncertainty in events. When the single selection policy is adopted (see the left column of Fig. 7), the average processing time is even lower than the one measured in the previous scenario.

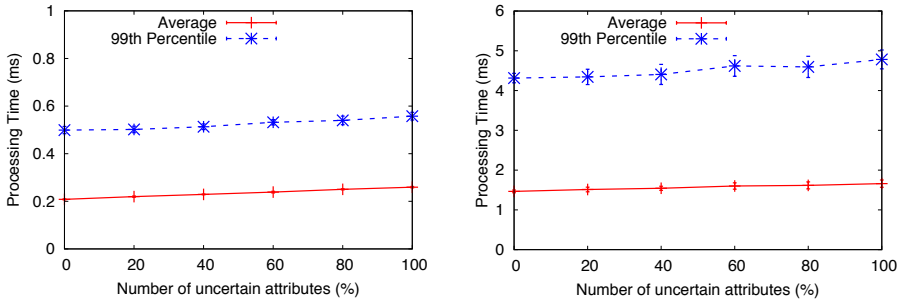
The average processing time decreases with the percentage of `Temp_i` events. Indeed, they simply need to be stored, while combination is performed on the arrival of a `LowOxygen_i`.

However, a high percentage of `Temp_i` increases the 99^{th} percentile of the processing time. In this setting, a large number of `Temp_i` events is generated and stored. This leads to a more complex processing when (infrequent) `LowOxygen_i` events enter the engine, triggering the evaluation of the combination constraint.

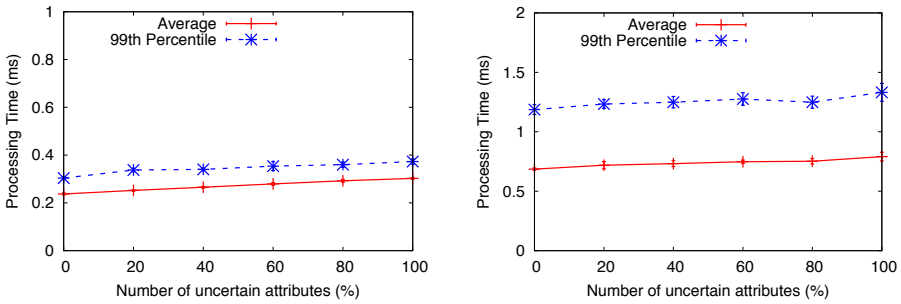
When considering a single selection policy, the engine needs to find only *one* `Temp_i` event to combine with every incoming `LowOxygen_i`. As soon as



(a) 10% of `LowOxygen_i` events



(b) 50% of `LowOxygen_i` events



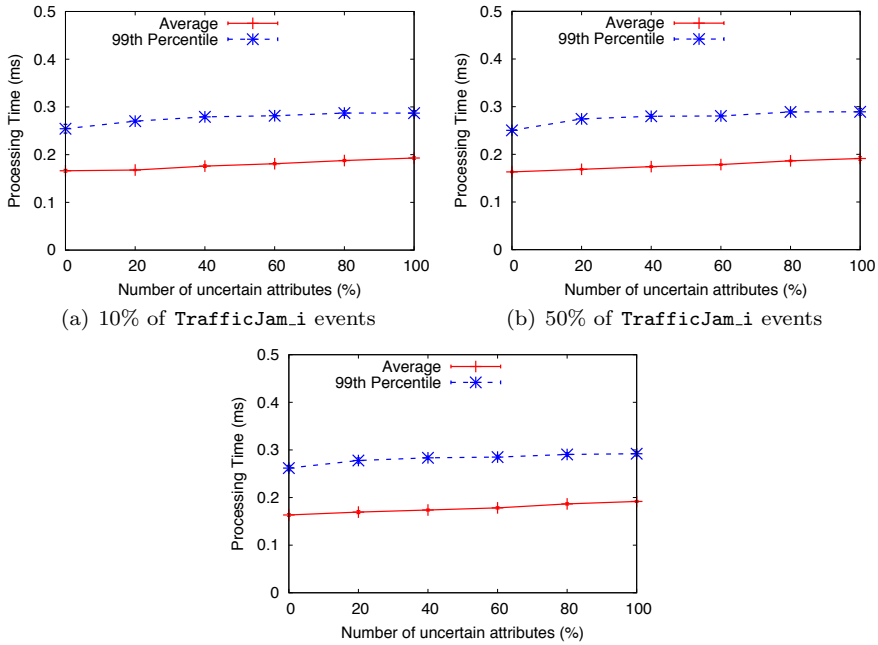
(c) 90% of `LowOxygen_i` events

Fig. 7: Combination: processing delay using the last-within (left) and each-within (right) operators

it finds one, it can safely stop processing. This is not possible in presence of a multiple selection policy (see the right column of Fig. 7), when *all* the `Temp_i` events need to be considered.

On the one hand, this implies higher processing times. On the other hand, the presence of a high number of `Temp_i` events becomes even more relevant. As the figure shows, both the average and the 99$^{\text{th}}$ percentile of the processing time significantly drop when lowering the percentage of `Temp_i`.

What is most important however, is that the impact of uncertainty in evaluating the combination operator is limited: in all the tests we performed,



(a) 10% of `TrafficJam_i` events

(b) 50% of `TrafficJam_i` events

(c) 90% of `TrafficJam_i` events

Fig. 8: Negation: processing delay

both when considering the average processing time and when considering the 99$^{\text{th}}$ percentile, the maximum overhead we measured was below 30%.

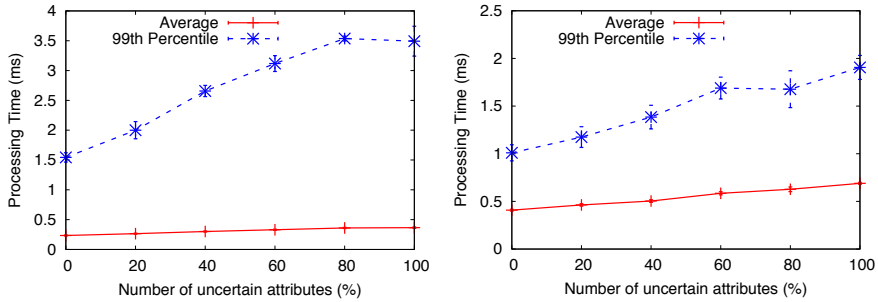
6.1.3 Negation

To evaluate the impact of uncertainty in presence of negation we deployed 1000 rules having the following structure:

```
define   TVS_Malfun_i()
from     Temp_i(km=$a and value>x) and
         not TrafficJam_i($a-10 < km < $a+10)
         within 5 min. from Temp_i
```

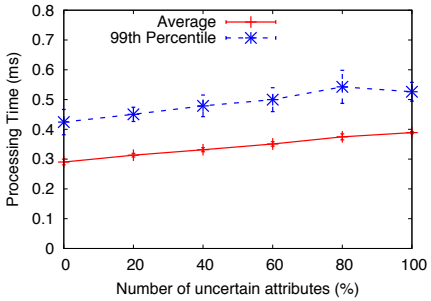
As in the previous scenario, we considered 10 different composite events (`TVS_Malfun_1`, ..., `TVS_Malfun_10`) and 100 different values for `x`, from 1 to 100. Both `km` and `value` attributes are uniformly distributed in the range 1–100. As in the previous case, we evaluate the engine with three different workloads, generating respectively 10%, 50%, and 90% of `TrafficJam_i` events (the remaining being `Temp_i`).

Fig. 8 shows the results we measured. They are similar to those obtained in evaluating composition with the `last-within` operator. Indeed, negation is implemented as a special form of composition under a single selection semantics; indeed the engine performs similar processing steps: it compares the



(a) 10% of `TrafficJam_i` events

(b) 50% of `TrafficJam_i` events



(c) 90% of `TrafficJam_i` events

Fig. 9: Aggregation: processing delay

values of `km` in different events and generates a single `TVS_Malfun_i` notification when no matching `TrafficJam_i` event is found.

In this setting, the presence of uncertainty has a minimal impact on the performance of the engine. Although the engine needs to compare every `TrafficJam_i` event received in the last 5 min upon detecting a `Temp_i` event this operation does not seriously affect the performance: the average processing time increases by about 0.03ms when moving from 0% to 100% of uncertain attributes, with a total overhead of less then 20%.

6.1.4 Aggregation

To evaluate the overhead of uncertainty when computing aggregates we deployed 1000 rules having the following structure:

```
define   TVS_Malfun_i()
from     LowOxygen_i() and x < $t=Avg(Temp_i().value
         within 5 min. from LowOxygen_i)
```

As in the previous scenarios, we built these rules by moving `i` in the range 1–10 and `x` in the range 1–100. Moreover, we considered three different workloads, generating respectively 10%, 50%, and 90% of `TrafficJam_i` events.

Fig. 9 shows the results we measured. In this case the overhead is greater than before, but it still acceptable. The maximum overhead we measured when

moving from 0% to 100% of uncertain attributes was below 75%, with an average processing time that remains always under 0.7ms in all the scenarios we tested.

A final note regards the 99^{th} percentile of the processing time. As Fig. 9 shows, it decreases as the percentage of `TrafficJam_i` events increases. Indeed, the CDP processing algorithm postpones the processing of aggregates until a valid sequence is found (in our case, until a `TrafficJam_i` event enters the engine). Consequently, a low number of `TrafficJam_i` implies infrequent aggregate evaluation; events of type `Temp_i` are simply accumulated, and contribute in lowering the average processing time. However, when a `TrafficJam_i` events arrives, the computation of aggregates starts and must consider a large number of `Temp_i` events (all those accumulated up to that point), which results in a high processing time that impacts the 99^{th} percentile.

6.1.5 Memory Overhead

As most existing CEP engines, to achieve good performance, T-Rex executes the processing of events entirely in main memory. Because of this, memory overhead represents a key criterion to evaluate our uncertainty model and implementation.

To analyze memory consumption, we focused on one of the scenarios discussed above, namely composition, and we measured the occupancy of memory in the two extreme cases in which (*i*) no attributes are uncertain and (*ii*) all the attributes are uncertain. More in particular, we repeated the experiments shown in Fig. 7a, which proved to be the most expensive in terms of processing time. We considered both a single selection scenario (using the `last-within` operator for event composition) and a multiple selection scenario (using the `each-within` operator).

In both cases, the maximum memory consumption we measured was below 15 MB. We can conclude that memory does not represent a bottleneck for the processing algorithm of T-Rex.

Most significantly, the overhead introduced by uncertainty management was negligible, always below 1% in all the tests we performed. Again, this is due to the specific architecture of T-Rex, which stores all incoming events only once and shares them across rules. Because of this, the more complex representation of attributes required to manage uncertainty does not significantly impact on the overall memory usage.

6.2 Accuracy

So far, our evaluation only targeted the overhead introduced by uncertainty management, i.e., its cost. Hereafter, we investigate the benefits of using uncertainty, i.e., the added value that a CEP user gets in receiving composite events that include uncertainty annotations.

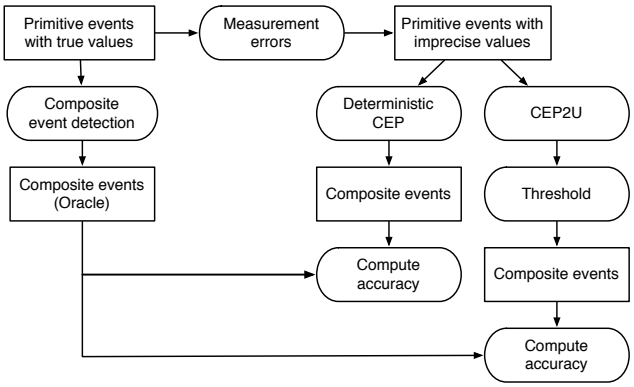


Fig. 10: Evaluation framework to test the accuracy of CEP2U

This is measured through the evaluation framework shown in Fig. 10. It starts by generating a random history of the primitive events that occur in the real world, i.e., those that include the *true* values of attributes, without any measurement error or uncertainty in data. These events are used to compute the actual occurrences of composite events as generated by a given Rule R. These are the composite events that happen in the real world and they represent our oracle. To evaluate the efficacy of CEP2U against this oracle, we introduce random measurement errors in the values of the original primitive events, according to the characteristics of our simulated sensors: these are the events we expect entering the CEP engine.

At this point, we use both a fully deterministic CEP engine (T-Rex) and our implementation of CEP2U to detect composite events starting from these primitive events. T-Rex only considers the actual values carried by events' attributes, while CEP2U also knows and uses the pdfs of the measurement errors they bring. T-Rex produces a set of composite events without any indication about uncertainty, while CEP2U also produces the probability of occurrence of each composite event. To compare the two sets of results, we introduce a threshold for CEP2U and filter out all the composite events whose probability of occurrence is lower than the threshold. Then, we study the accuracy of CEP2U while changing such a threshold. In particular, to calculate the accuracy of our CEP engines (i.e., T-Rex and CEP2U) we refer to the results coming from the oracle, i.e., the *true* composite events, denoting:

`TP` the number of true positives, i.e., the number of composite events that are detected both by the oracle and by the CEP engine under examination;
`TN` the number of true negatives, i.e., the number of composite events that are neither detected by the oracle nor by the CEP engine;
`FP` the number of false positives, i.e., the number of composite events that are detected by the engine but not by the oracle;

`FN` the number of false negatives, i.e., the number of composite events that actually occurred (according to the oracle) but were not detected by the engine.

Using these definitions, the accuracy of each of the two engines under examination can be computed as follows:

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN}$$

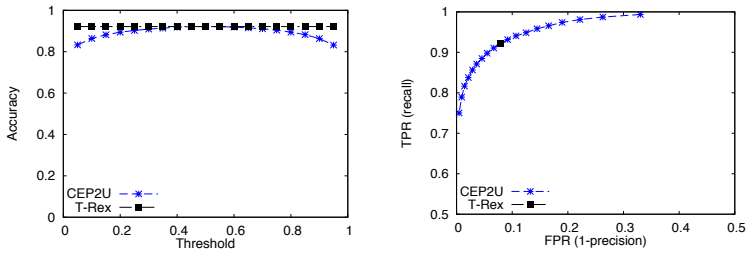
Intuitively, an accuracy of 1 means that the engine works perfectly (there are neither false positives `FP` nor false negatives `FN`). Conversely, the accuracy decreases as more errors are generated. In the following, we analyze and compare the accuracy of T-Rex and CEP2U using three different workloads:

- *Selection* involves one rule that predicates on a single primitive event using a selection constraint;
- *Combination* involves one rule that predicates on two primitive event using a combination constraint;
- *Hierarchy* involves two rules. Each rule predicates on one event using a selection constraint. The output of the first rule is used as an input for the second rule, thus building a *hierarchy* of events.

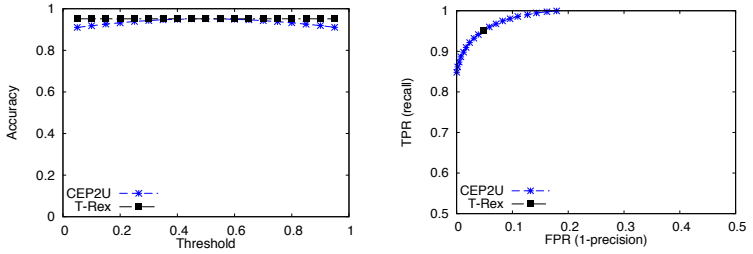
In all our tests we consider the measurement errors (i.e., the sensors) to have a normal distribution $N(0, 1)$. For the selection scenario, we also consider the case of a uniform distribution $U(-1, 1)$.

Fig. 11 shows the results we measured. The column on the left shows the accuracy for both T-Rex and CEP2U, while the column on the right shows the Receiver Operating Characteristic (ROC) curve for CEP2U and how it compares with the T-Rex accuracy. The ROC curve shows the relation between the rate of false positives (`FPR`) and the rate of true positives (`TPR`) while changing the threshold. Intuitively, an engine that chooses at random (random guessing about the occurrence or non occurrence of a composite event) would lie on the diagonal of the plot. A perfect engine would always lie in the upper left corner. The ROC curve shows how a human expert can tune the threshold of acceptance to balance between recall (i.e., minimize the number of false negatives) and precision (i.e., minimize the number of false positives).

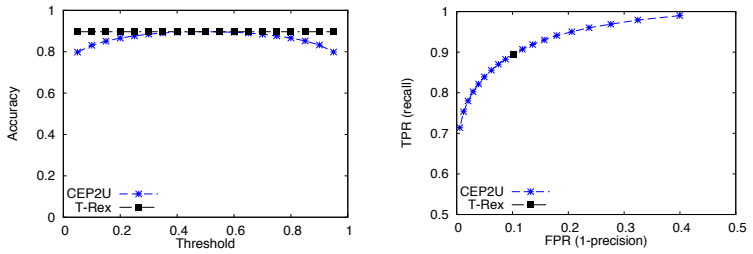
In all the scenarios we tested, CEP2U provides a level of accuracy higher than 0.8. Such value reaches its maximum with a threshold of 0.5, when CEP2U behaves as the fully deterministic T-Rex engine. However, CEP2U offers a much higher degree of flexibility: indeed, users receive composite events annotated with their probability of occurrence and are free to decide when to accept and when to discard them, i.e., users can set their own threshold of acceptance. This is useful in all those scenarios in which the cost of a false positive and the cost of a false negative are not equal: for example, in a nuclear power plant, it is certainly worth considering warning events even if they carry a low probability of occurrence. This is something that a deterministic engine does not provide, always balancing the cost of false positives and that



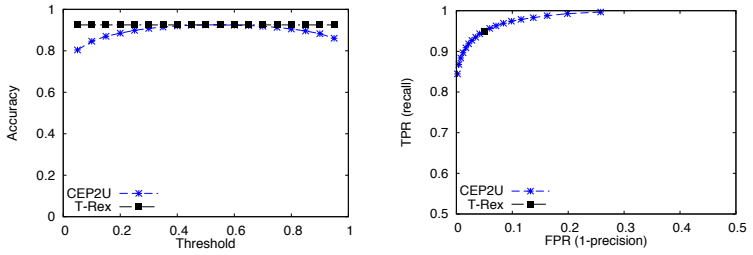
(a) Selection (Gaussian Error)



(b) Selection (Uniform Error)



(c) Combination



(d) Hierarchy

Fig. 11: Accuracy and ROC graphs for different kinds of rules

of false negatives and ignoring the composite events that implicitly have a low probability of occurrence.

Looking at the right column of Fig. 11 we also observe that CEP2U offers a good balance between precision and recall while changing the threshold of acceptance.

Two additional considerations emerge from the results in Fig. 11. First, as expected, the distribution of imprecision in data has a significant impact on accuracy: this is clearly visible when comparing a normally distributed error (Fig. 11a) and a uniformly distributed error (Fig. 11b). Second, in evaluating the approach of CEP2U in dealing with hierarchies of events (Fig. 11d), we may observe that the choice of considering all composite events that re-enter the processing engine as having a probability of 1, CEP2U tend to favorite false positives against false negatives. Because of this, the graph in Fig. 11d, left becomes slightly asymmetric. Nevertheless, the level of accuracy remains higher that 0.8, even when considering a very low threshold. While our approach for dealing with hierarchies of events was mainly motivated by performance reasons, it still proved to provide high accuracy, as also proved by the ROC graph in Fig. 11d.

6.3 Uncertainty in Rules

This section measures the time required to evaluate the BN associated to a rule. As we mentioned in Section 4.2.2, this step occurs entirely at rule design time[8].

In particular, we considered Rule R1, we let CEP2U create the corresponding BN, and we enriched it by adding new nodes. We measured the evaluation time while changing the number of additional nodes introduced in the BN. We repeat each experiment 1000 times with different randomly generated probability tables for the nodes. We compute the average value and its 99^{th} confidence interval.

Fig. 12 shows the results we obtained: even when considering a large number of additional nodes –i.e., of external factors contributing to the probability of occurrence of the composite event– the evaluation of the BN only takes less than 0.5ms. Moreover, this value is even lower when considering a reduced number of nodes, which we expect to be a common case in most applications.

We can conclude that the amount of processing required to evaluate the uncertainty of rules is negligible: after every rule update, updating the probability of the composite event is performed in sub-millisecond time.

6.4 Discussing CEP2U Performance

Our relatively long experience in evaluating CEP systems [15,16] convinced us of the difficulty of finding general answers, due to the huge number of parameters that may impact performance. To address this issue, in this section we evaluated the processing overhead coming from uncertainty, by separately considering the various operators provided by CEP languages. For each of

[8] Notice that to capture uncertainty in rules, after evaluating BNs at rule design time, we have to propagate the calculated value to the composite events; a step that happens at run-time but with no measurable impact on performance.

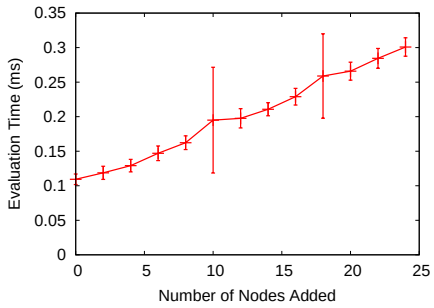


Fig. 12: Time to evaluate an enriched BN

them, we focused on a few scenarios chosen to minimize the effects of T-Rex specific optimization (e.g., early pruning of constraint evaluation by exploiting event types) and to maximize the amount of processing required to support uncertainty.

Even considering this challenging scenarios, the maximum overhead we measured when introducing uncertainty was relatively low: in all the tests we performed the overall processing time increases by less than 50%, the only exception being the simple selection scenario.

During our tests, we assumed the measurement error to be normally distributed. This is a reasonable assumption in most application fields. Moreover, normal distributions are often adopted as an approximation when providing an analytic distribution is impossible or computationally expensive.

By separating uncertainty in events from uncertainty of rules, CEP2U enables the computation of the latter at design time, when a new rule is deployed or modified. Moreover, by exploiting a separate BN for each rule, it reduces the processing effort required to evaluate the impact of external factors on the triggering probability of a rule. As shown in our tests, even when the model used to represent a rule takes into account a large number of factors, the evaluation of rule uncertainty using a BN can be performed in sub-ms time.

Furthermore, despite CEP2U demands for a more complex representation of events and attributes, this does not significantly impact on memory consumption.

Finally, CEP2U combines good performance with a high level expressiveness and accuracy. By annotating composite events with their probability of occurrence, it enables CEP users to flexibly select which event are relevant and which can be ignored, choosing the balance between false positives and false negatives that better fit the application domain.

In summary we can conclude that the key choices we made in designing CEP2U resulted in a model of uncertainty for CEP that is expressive and effective but also efficient.

7 Related Work

This section reviews related work. First of all, we present CEP systems, focusing on the rule definition languages they adopt and on the processing algorithms they provide. This aims at showing the general applicability of CEP2U. Second, we discuss existing models and solutions to deal with uncertainty in CEP.

7.1 Complex Event Processing

The last few years have seen an increasing interest around Complex Event Processing, with several CEP systems being proposed both from academia and industry [29,21]. The interested reader can find a detailed study of the field in [17], where we analyze and compare in great detail more than 35 systems. Despite all existing solutions have been designed to accomplish the same goal, i.e., to timely process large amount of flowing data, they present different data models and rule definition languages, as well as processing algorithms and system architectures.

7.1.1 Data Models and Rule Definition Languages

The data model determines the way each system models and interprets incoming information: as mentioned in Section 2, the database community gave birth to *Data Stream Management Systems (DSMSs)* [8] to process generic information streams. On the other hand, the community working on event-based systems focused on a form of data —event notifications— with a very specific semantics, in which the time (of occurrence) plays a central role [33].

In practice, the data model a system adopts significantly affects the structure of the rule definition language it uses. DSMSs usually rely on languages derived from SQL, which specify how incoming data have to be transformed, i.e., selected, joined together, and modified, to produce one or more output streams. Processing happens in three steps [6]: first, *Stream-to-Relation* (S2R) operators (also known as *windows*) select a portion of a stream to implicitly create traditional database tables. The actual computation occurs on these tables, using *Relation-to-Relation* (R2R) operators —mostly standard SQL operators. Finally, *Relation-to-Stream* (R2S) operators generate new streams from tables, after data manipulation. Despite several extensions have been proposed [20,43,35], they all rely on the general processing schema described above.

At the opposite side of the spectrum are languages that were explicitly designed to capture composite events from primitive ones [21]. They interpret messages flowing into the system as notifications of events occurred in the observed world at a specific time, and they define how composite events result from primitive ones. The TESLA language we used to exemplify our model of uncertainty belongs to this second class. TESLA represents a good test

case for CEP2U because of its expressiveness. Indeed, languages belonging to this second class often trade simplicity and performance for expressiveness: for example, some languages force sequences to capture only adjacent events [11]; negations are rarely supported [28,11] or they cannot be expressed through timing constraints [2]; other widespread limitations are the lack of a full-fledged iteration operator (Kleene+ [23]), to capture a priory unbounded repetitions of events, and the lack of processing capabilities for computing aggregates. Finally, none of these languages allows to define the selection and consumption policies rule by rule, as TESLA does.

The languages that present more similarities with TESLA are the language for complex event detection presented in [36], Sase+ [23,2], Amit [1], and Etalis [5,4].

Finally, there are commercial systems [44,37,35] that try to combine the two aforementioned approaches, offering hybrid languages that allow both SQL-like processing and pattern detection in a single framework.

As we already mentioned in Section 2, CEP2U is agnostic w.r.t. the peculiarities of the rule language adopted. As far as the uncertainty in events is concerned, CEP2U supports selection of single events, combination of multiple events according to the attributes they carry, negation, arbitrary computation over the content of events, and propagation of the results of computation to the composite events generated. These operators cover all the processing capabilities of both the languages based on relational operators and the languages based on patterns. In the first case, selection and combination of events is performed through the select and join operators defined in the relational algebra. In the second case they are defined through pattern matching, using logic operators (like conjunctions and disjunctions) often complemented with timing constraints, as in the `*-within` TESLA operators. Both kinds of languages allow some form of computation using the values stored in incoming events; finally, both kinds of languages generate new events as a result of their processing.

Moreover, CEP2U considers the uncertainty of rules outside the rule definition language, using Bayesian Networks to model the causal relations that bind together primitive and composite events, this way it straightforwardly applies to both classes of languages.

7.1.2 Processing Algorithms

The language used to specify rules significantly influences the processing algorithms adopted and their performance. DSMSs usually translate the set of deployed rules into a query plan composed of primitive operators (e.g., selection, projection, join) that transform the input streams into one or more output streams.

Systems that offer native support for pattern detection often implement algorithms based on automata [28,11,2,20,36], where the processing is performed incrementally, as new primitive events enter the engine. We adopted a similar approach in our first implementation of T-Rex [15]. The CDP algorithm

adopted in this paper takes a different approach: it stores primitive events and delays the processing as much as possible. In [16] we show the advantages, in terms of processing delay and throughput, of CDP over automata-based algorithms; moreover, we show how CDP can be easily parallelized to take advantage of multi-core hardware.

As our implementation in T-Rex demonstrates, adapting an existing CEP engine to support CEP2U is relatively easy. On the one hand, the uncertainty of rules is completely delegated to Bayesian Networks: this only requires a new module for translating rules into the corresponding BN; all the tools used for editing and evaluating BNs can be re-used without any modification. On the other hand, dealing with the uncertainty in events only requires a modification of the functions used to evaluate constraints and combine uncertain data. These changes are language and algorithm-specific but, as we show in this paper considering TESLA and CDP, they involve relatively simple modifications to the engine.

As a final note, we observe that in some specific cases, the changes mentioned above could impact ad-hoc data structures and algorithms that cannot be easily adapted to fit uncertain data. We provided an example of this situation in Section 5, when discussing the need of completely re-writing the T-Rex `Static Index` component. The performance overhead of these changes may vary from system to system.

7.2 Models and Solutions for Uncertainty

Despite uncertainty handling has been recognized as one of the most critical and relevant aspects in the area of CEP [7,17], it still remains an open issue. Only a few solutions have been proposed, and most of them are tailored to a specific application domain.

To the best of our knowledge, the first model proposed for dealing with uncertainty in CEP is described in [47]. This model has been extended in [48], where the authors introduce a general framework for CEP in presence of uncertainty. It captures the sources of uncertainty we consider in this paper —i.e., the uncertainty in events and the uncertainty in rules— and adopts Bayesian Networks to model both of them. Differently from our approach, it creates a single BN, including all the possible events of interest; such a BN is continuously updated at run-time, as new primitive events are observed.

Since the BN is used to capture the uncertainty in events, it is significantly more complex than those generated by CEP2U: indeed, its probability tables must include all possible attribute values. Moreover, the BN cannot be modified to model external factors that are not captured by rules. As far as performance is concerned, the proposed methodology requires a (partial) reconstruction of the BN every time a new primitive event e is detected. The complexity of such reconstruction is exponential in the number of nodes influenced by the arrival of e. This significantly impacts processing time: the

system described in [48] produces a maximum throughput of less than 1000 events/s, decreasing to hundreds, or even tens of events in many scenarios.

Further details about the framework are presented in [49]. An extended evaluation of the proposed approach confirms the performance results described in [48].

A similar approach is presented in [38]. The authors put a great emphasis on the processing algorithms proposed and on their performance. More in particular, they focus on a reduced set of operators —a subset of the rule language offered by Cayuga [11]— and propose the integration of probabilistic evaluation into an automata-based processing algorithm. Interestingly, the authors not only focus on real time processing, but also address stored data and propose pre-processing and analysis techniques to speed up queries over probabilistic data. In comparison, CEP2U can be applied to more expressive languages, while introducing low overheads at run-time. Moreover, CEP2U enables finer grained modeling of uncertainty in rules through BNs.

In [19], the authors present a model to capture and propagate the uncertainty of primitive events in Data Steam Management Systems (DSMSs). Similarly to our approach, uncertainty in events is modeled and processed using the probability theory. Differently from CEP2U, this solution is not capable of modeling uncertainty in rules. Moreover, the proposed model has not been implemented in a running system, making it impossible to evaluate its performance and overhead.

A tutorial has been presented in the DEBS (Distributed Event Based Systems) 2012 conference, entirely dedicated to event processing under uncertainty [7]. The tutorial points out the need for uncertainty handling in event processing and proposes a classification of the possible sources of uncertainty that is similar to the one we adopt in this paper. Moreover, the authors acknowledge the need for modeling and propagation of uncertainty, and propose probability theory as a possible mathematical foundation to accomplish these tasks.

Markov Logic Networks (MLNs) [39] represent an effective formalism to deal with uncertainty. Briefly, MLNs incorporate both hard logical statements expressed as first order logic formulae as well as a probabilistic reasoning engine in a unifying mathematical framework. More precisely, a MLN is a first order knowledge base in which uncertainty is modelled by weights attached to each first order logic formula. The knowledge base is then used as a template for constructing a Markov network from which probabilistic inference is computed. Being based on first order logic, MLNs are extremely expressive and, from an abstract viewpoint, they could be used to reason under uncertainty in the scenarios described in this paper. Indeed, the event patterns of interest of a CEP application could be encoded in first order logic and existing MLN reasoners may be used for event processing under uncertainty.

However, in practice, two crucial aspects make this choice impractical for real-world CEP applications and justify instead the adoption of CEP2U.

First, encoding event patterns in first order logic is a difficult, time consuming and tedious task. Domain experts may not have the appropriate back-

ground to effectively accomplish this task which is, per se, already difficult for engineers with expertise in logical formalisms. CEP2U builds instead on top of the rule-based paradigm of existing CEP engines and preserves the same high level declarative philosophy explicitly conceived to capture complex event patterns. More specifically, it does not require any particular prior engineering knowledge from domain experts. This may seem a naïve advantage but, in our opinion, it makes the difference from a theoretical approach to a solution that can be easily applied in the field.

Secondly, reasoning under uncertainty with MLNs is an NP-complete problem [39]. It requires a computational effort that is incompatible with the throughput constraints imposed by typical CEP application domains, which require to handle up to thousands of events per second with hundreds of rules deployed into the engine. Differently from MLNs, CEP2U targets at preserving the high level of performance typically offered by existing CEP engines, augmenting at the same time their expressiveness to support uncertainty. As anticipated in Section 1, simplicity as well as efficiency are two aspects that drove the inception and design of CEP2U since they reflects two crucial requirements of real-world CEP applications.

Despite the limitations discussed above, MLNs represent a valid tool to reason under uncertainty that has been successfully used in specific domains characterized by different requirements with respect to CEP applications. For the sake of completeness, we briefly exemplify hereafter some existing works based on MLNs. The community of visual event and activity recognition proposed several solutions for event detection and recognition. Among these existing works it's worth to mention the work by Tran et al. [45] that addresses the problem of visual event recognition in visual surveillance. In their approach the domain knowledge is represented using first order logic statements in which both negation and disjunction are allowed, while uncertainty of primitive event detection is represented using detection probabilities. Logical statements and probabilities are combined into a single framework using MLNs. Analogously, the works by Kembhavi et al. [27] and Morariu et al. [32] discuss two frameworks for activity recognition in structured scenarios based on MLNs. Another relevant existing approach for activity recognition is described in [24]. In this work the authors propose to use MLNs as a statistical relational framework for activity recognition in ambient assisted living environments. Finally it's worth to mention the work by Biswas [9], which introduces instead a first-order probabilistic model that combines multiple cues to classify human activities from video data that relies on dynamic MLNs [40].

8 Conclusions

In this paper, we presented *CEP2U* (*Complex Event Processing under Uncertainty*), a novel model for dealing with uncertainty in CEP that provides a valuable combination of expressiveness, efficiency, and ease of use.

We applied CEP2U to TESLA, showing how it seamlessly supports and integrates with the typical operators offered by a modern CEP rule language. It is our claim that our approach significantly reduces the complexity of the uncertainty management problem, by allowing engineers and domain experts to easily and separately capture the uncertainty coming from incoming event notifications and that inherent in rules.

Our experience in implementing CEP2U on top of T-Rex shows the limited effort required to integrate CEP2U with an existing CEP engine, while the detailed performance analysis of CEP2U we provided in this paper, shows how it introduces a reduced overhead: less than 50% in all tests we run.

Future work includes investigating further sources of uncertainty and the application to a real case study. Furthermore, we plan to consider learning mechanisms to automatically generate rules from historical data analysis [31]. This could also help domain experts in determining some critical parameters related to uncertainty (e.g., tune the minimum probability for considering events based on past occurrences).

To conclude, we believe that uncertainty managements constitutes a critical yet necessary component in most event processing applications. We hope that CEP2U could represent a starting point to promote further investiagations in this area.

Acknowledgment

This research has been funded by the European Commission, Programme IDEAS-ERC, Project 227977-SMScom and Programme FP7-PEOPLE-2011-IEF, Project 302648-RunMore and by the Dutch national program COMMIT.

References

1. Adi, A., Etzion, O.: Amit - the situation manager. The VLDB Journal **13**(2), 177–203 (2004). DOI http://dx.doi.org/10.1007/s00778-003-0108-y
2. Agrawal, J., Diao, Y., Gyllstrom, D., Immerman, N.: Efficient pattern matching over event streams. In: SIGMOD, pp. 147–160. ACM, New York, NY, USA (2008). DOI http://doi.acm.org/10.1145/1376616.1376634
3. Aguilera, M.K., Strom, R.E., Sturman, D.C., Astley, M., Chandra, T.D.: Matching events in a content-based subscription system. In: Proceedings of the eighteenth annual ACM symposium on Principles of distributed computing, PODC '99, pp. 53–61. ACM, New York, NY, USA (1999). DOI 10.1145/301308.301326
4. Anicic, D., Fodor, P., Rudolph, S., Stuhmer, R., Stojanovic, N., Studer, R.: A rule-based language for complex event processing and reasoning. In: P. Hitzler, T. Lukasiewicz (eds.) Web Reasoning and Rule Systems, *Lecture Notes in Computer Science*, vol. 6333, pp. 42–57. Springer Berlin / Heidelberg (2010)
5. Anicic, D., Fodor, P., Rudolph, S., Stuhmer, R., Stojanovic, N., Studer, R.: Etalis: Rule-based reasoning in event processing. In: S. Helmer, A. Poulovassilis, F. Xhafa (eds.) Reasoning in Event-Based Distributed Systems, *Studies in Computational Intelligence*, vol. 347, pp. 99–124. Springer Berlin / Heidelberg (2011)
6. Arasu, A., Babu, S., Widom, J.: The cql continuous query language: semantic foundations and query execution. The VLDB Journal **15**(2), 121–142 (2006)

7. Artikis, A., Etzion, O., Feldman, Z., Fournier, F.: Event processing under uncertainty. In: DEBS (2012)
8. Babcock, B., Babu, S., Datar, M., Motwani, R., Widom, J.: Models and issues in data stream systems. In: PODS, pp. 1–16. ACM, New York, NY, USA (2002)
9. Biswas, R., Thrun, S., Fujimura, K.: Recognizing activities with multiple cues. In: Workshop on Human Motion, pp. 255–270 (2007)
10. BOOST: BOOST C++ Libraries: Math Toolkit (2012). Http://www.boost.org/doc/libs/1_49_0/libs/math/doc/sf_and_dist/html/
11. Brenna, L., Demers, A., Gehrke, J., Hong, M., Ossher, J., Panda, B., Riedewald, M., Thatte, M., White, W.: Cayuga: a high-performance event processing engine. In: SIGMOD, pp. 1100–1102. ACM, New York, NY, USA (2007)
12. Broda, K., Clark, K., 0002, R.M., Russo, A.: Sage: A logical agent-based environment monitoring and control system. In: AmI, pp. 112–117 (2009)
13. Chakravarthy, S., Krishnaprasad, V., Anwar, E., Kim, S.K.: Composite events for active databases: Semantics, contexts and detection. In: Proceedings of the 20th International Conference on Very Large Data Bases, VLDB '94, pp. 606–617. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1994)
14. Cugola, G., Margara, A.: Tesla: A formally defined event specification language. In: DEBS, pp. 50–61 (2010)
15. Cugola, G., Margara, A.: Complex event processing with t-rex. Journal of Systems and Software **85**(8), 1709 – 1728 (2012). DOI 10.1016/j.jss.2012.03.056
16. Cugola, G., Margara, A.: Low latency complex event processing on parallel hardware. Journal of Parallel and Distributed Computing **72**(2), 205 – 218 (2012). DOI 10.1016/j.jpdc.2011.11.002
17. Cugola, G., Margara, A.: Processing flows of information: From data stream to complex event processing. ACM Comput. Surv. **44**(3), 15:1–15:62 (2012). DOI 10.1145/2187671.2187677
18. Demers, A.J., Gehrke, J., Hong, M., Riedewald, M., White, W.M.: Towards expressive publish/subscribe systems. In: EDBT, pp. 627–644 (2006)
19. Diao, Y., Li, B., Liu, A., Peng, L., Sutton, C., Tran, T.T.L., Zink, M.: Capturing data uncertainty in high-volume stream processing. In: CIDR 2009, Fourth Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, January 4-7, 2009, Online Proceedings (2009)
20. Esper, http://esper.codehaus.org/ (2012)
21. Etzion, O., Niblett, P.: Event Processing in Action. Manning Publications Co. (2010)
22. Event zero, http://www.eventzero.com/solutions/environment.aspx (2012)
23. Gyllstrom, D., Agrawal, J., Diao, Y., Immerman, N.: On supporting kleene closure over event streams. In: ICDE, pp. 1391–1393 (2008)
24. Helaoui, R., Niepert, M., Stuckenschmidt, H.: Recognizing interleaved and concurrent activities: A statistical-relational approach. In: PerCom, pp. 1–9 (2011)
25. Jensen, F.: An introduction to Bayesian networks, vol. 36. UCL press London (1996)
26. Jensen, F.: An introduction to Bayesian networks, vol. 36. UCL press London (1996)
27. Kembhavi, A., Yeh, T., Davis, L.S.: Why did the person cross the road (there)? scene understanding using probabilistic logic models and common sense reasoning. In: ECCV (2), pp. 693–706 (2010)
28. Li, G., Jacobsen, H.A.: Composite subscriptions in content-based publish/subscribe systems. In: Middleware, pp. 249–269. Springer-Verlag New York, Inc. (2005)
29. Luckham, D.C.: The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (2001)
30. Margara, A.: Combining expressiveness and efficiency in a complex event processing middleware. Ph.D. thesis, Politecnico di Milano (2012)
31. Margara, A., Cugola, G., Tamburrelli, G.: Towards automated rule learning for complex event processing (2013). Technical Report
32. Morariu, V.I., Davis, L.S.: Multi-agent event recognition in structured scenarios. In: CVPR, pp. 3289–3296 (2011)
33. Mühl, G., Fiege, L., Pietzuch, P.: Distributed Event-Based Systems. Springer-Verlag New York, Inc., Secaucus, NJ, USA (2006)

34. Netica: Netica API (2012). Http://www.norsys.com/netica_api.html
35. Oracle cep. http://www.oracle.com/technologies/soa/complex-event-processing.html (2011)
36. Pietzuch, P.R., Shand, B., Bacon, J.: A framework for event composition in distributed systems. In: Proceedings of the ACM/IFIP/USENIX 2003 International Conference on Middleware, Middleware '03, pp. 62–82. Springer-Verlag New York, Inc., New York, NY, USA (2003)
37. Progress-Apama: http://web.progress.com/it-need/complex-event-processing.html (2011). Visited Nov. 2011
38. Ré, C., Letchner, J., Balazinksa, M., Suciu, D.: Event queries on correlated probabilistic streams. In: Proceedings of the 2008 ACM SIGMOD international conference on Management of data, SIGMOD '08, pp. 715–728. ACM, New York, NY, USA (2008). DOI 10.1145/1376616.1376688
39. Richardson, M., Domingos, P.: Markov logic networks. Machine learning **62**(1-2), 107–136 (2006)
40. Sanghai, S., Domingos, P., Weld, D.: Learning models of relational stochastic processes. In: Machine Learning: ECML 2005, pp. 715–723. Springer (2005)
41. Schultz-Møller, N.P., Migliavacca, M., Pietzuch, P.R.: Distributed complex event processing with query rewriting. In: DEBS, pp. 4:1–4:12 (2009)
42. Srivastava, U., Widom, J.: Flexible time management in data stream systems. In: PODS '04, pp. 263–274. ACM, New York, NY, USA (2004). DOI http://doi.acm.org/10.1145/1055558.1055596
43. Streambase, http://www.streambase.com/ (2011)
44. Tibco: Tibco BusinessEvents. http://www.tibco.com/software/complex-event-processing/businessevents/default.jsp (2011). Visited Nov. 2011
45. Tran, S.D., Davis, L.S.: Event modeling and recognition using markov logic networks. In: ECCV (2), pp. 610–623 (2008)
46. Wang, F., Liu, P.: Temporal management of rfid data. In: VLDB, pp. 1128–1139. VLDB Endowment (2005)
47. Wasserkrug, S., Gal, A., Etzion, O.: A model for reasoning with uncertain rules in event composition systems. In: Proceedings of the 21st Annual Conference on Uncertainty in Artificial Intelligence, pp. 599–606 (2005)
48. Wasserkrug, S., Gal, A., Etzion, O., Turchin, Y.: Complex event processing over uncertain data. In: Proceedings of the second international conference on Distributed event-based systems, DEBS '08, pp. 253–264. ACM, New York, NY, USA (2008). DOI 10.1145/1385989.1386022
49. Wasserkrug, S., Gal, A., Etzion, O., Turchin, Y.: Efficient processing of uncertain events in rule-based systems. IEEE Trans. on Knowl. and Data Eng. **24**(1), 45–58 (2012). DOI 10.1109/TKDE.2010.204