IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received May 20, 2019, accepted June 12, 2019, date of publication June 19, 2019, date of current version July 18, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2923953

Managing Measurement and Occurrence Uncertainty in Complex Event Processing Systems

NATHALIE MORENO[1], MANUEL F. BERTOA[1], LOLI BURGUEÑO[2,3], AND ANTONIO VALLECILLO[1]

[1]Departamento de Lenguajes y Ciencias de la Computación, Universidad de Málaga, 29071 Málaga, Spain
[2]IN3, Open University of Catalonia, 08035 Barcelona, Spain
[3]Institut LIST, CEA, Université Paris-Saclay, 91120 Paris, France

Corresponding author: Nathalie Moreno (moreno@lcc.uma.es)

This work was supported by the Spanish Research Project under Grant TIN2014-52034-R, Grant TIN2016-75944-R, and Grant PGC2018-094905-B-I00.

ABSTRACT Complex event processing (CEP) is a powerful technology for analyzing streams of real-time events, coming from different sources, and for extracting conclusions from them. In many situations, these events are not free from uncertainty, due to either unreliable data sources and networks, measurement uncertainty, or inability to determine whether an event has actually happened or not. This paper presents a proposal for incorporating and managing different kinds of uncertainty that may happen in both events and rules of the CEP systems. We provide a library that enables the representation and propagation of uncertain values, which can be efficiently integrated with the existing CEP languages and engines to deal with uncertainty, and we show how the treatment of uncertainty can be smoothly added to two of them: Esper and Apache Flink. Five applications coming from various domains serve to evaluate the proposal and to analyze its performance and accuracy. The results show that the overhead introduced by the treatment of uncertainty is not high and good precision and recall are achieved.

INDEX TERMS Complex event processing, measurement uncertainty, stream processing.

I. INTRODUCTION

Complex Event Processing (CEP) systems are being widely adopted as they provide effective means for processing and analyzing the steadily growing number of information sources that continuously produce and offer data in many applications of interest. Examples of such applications include monitoring systems for critical infrastructures [1], environmental monitoring [2]–[4], stock market analysis [5], network analysis and surveillance [6], maritime vessels trajectory monitoring [7], and social media data aggregation [8], [9]. One domain where CEP is particularly relevant is the Internet of Things (IoT) [10], [11], where applications should process and react to events arriving from various kinds of sources including wireless sensor networks, RFID devices, GPS, etc.

In a nutshell, CEP is a stream-processing system for analyzing and correlating streams of data about real-time events that happen in a system, and deriving conclusions from them [12]–[15]. A distinguishing feature of CEP, not present in most stream-processing systems, is that it permits defining complex events on top of simple events (raw data), to identify meaningful circumstances and to respond to them as quickly as possible. Different domain-specific languages called Event Processing Languages (EPLs) and engines for processing events currently exist, such as Esper,[1] Apache Flink,[2] Microsoft Azure Stream Analytics,[3] or Tesla [16]. CEP programs are usually composed by a set of rules. Each rule defines a pattern and creates a complex event every time the pattern matches the events in the stream.

One particular aspect that cannot be neglected when dealing with physical systems and networks, is that the events are not free from uncertainty [17]. It may be due to different causes, including unreliable data sources and networks, measurement uncertainty, or the inability to determine whether an event has actually happened or not. Our CEP application code may also have some associated uncertainty, when we are not 100% confident on the rules. Some authors, e.g.

The associate editor coordinating the review of this manuscript and approving it for publication was Saeid Nahavandi.

[1]http://www.espertech.com/esper/
[2]https://flink.apache.org/
[3]https://azure.microsoft.com/en-us/services/stream-analytics/

[18]–[21], have addressed these issues, using different techniques and covering some of the aspects related to the representation, management and propagation of uncertainty. However, their proposals usually suffer from two main limitations. First, they normally focus on specific aspects of uncertainty in CEP systems (uncertainty in event attributes, event occurrence, event timestamps or rules), with only partial coverage of all these problems (cf. [22]). Second, propagation of attributes' uncertainty through operations is manually done, which poses the burden of such a cumbersome and error prone task on the system modeler.

This paper provides a classification of the different types of uncertainty that may happen in CEP systems, and discusses how to incorporate them into CEP events and rules. For this, we use a Java library developed in a previous work [23] with the intent to support an extension of the Object Constraint Language (OCL) [24] and the Unified Modeling Language (UML) [25] primitive datatypes. With this library, we are able to represent uncertain values and transparently deal with measurement uncertainty. Here, we reuse this library with a different purpose: we show how it can be used to address measurement and occurrence uncertainty in the events, in the values of their attributes (including their timestamps), and in the confidence of rules. We use two different CEP engines, namely Esper and Apache Flink, to apply and evaluate the results of our approach according to different dimensions: correctness, performance, accuracy and applicability. We have applied our approach to five different case studies, each one exhibiting different characteristics.

This paper is an extension of [26], where we sketched the initial ideas behind our approach. We have extended the original paper in several directions. First, we have separated our approach from the EPL of choice, by encapsulating the treatment of uncertainty in a Java library. In this respect, we show how our proposal can be incorporated into different EPLs, namely Esper and Apache Flink. Secondly, we have extended our proposal to deal with the uncertainty of dependent events, as well as with events that have been registered but have not happened in reality (false positives) and lost events (false negatives). Thirdly, in this paper, we provide a more in-depth evaluation of our approach, including an assessment of its correctness, performance, accuracy and applicability using five CEP applications from different domains.

The structure of the paper is as follows. First, Sect. II briefly introduces CEP systems and their main features, as well as the basic uncertainty issues that may happen in CEP. A running example is used to illustrate the CEP basic concepts and mechanisms. Then, Sect. III describes our extension for dealing with uncertainty in CEP systems, and the implementation that we have developed for the Esper and Apache Flink CEP engines using our library for uncertainty. The evaluation we have conducted on our proposal is described in Sect. IV, including a set of case studies of different nature that we have used to evaluate our approach as well as the limitations we have found. Sect. V compares our work with other existing proposals for dealing with uncertainty in CEP systems, discussing their benefits and limitations. Finally, Sect. VI concludes and outlines some future lines of work.

II. BACKGROUND

A. COMPLEX EVENT PROCESSING

CEP [13], [14] is a form of Information Processing [12] whose goal is the definition and detection of situations of interest, from the analysis of low-level event notifications [27]. According to the Event Processing Technical Society [28], the term *simple event* refers to the low-level primitive event occurrences, and *complex event* to those that summarize, represent, or denote a set of other events. Complex events are derived by rules that define the relevant *patterns* of (simple or other complex) events, their contents, and their temporal relations.

Although several CEP systems and languages exist [16], [29], [30], from a user's point of view they all share the same basic concepts, mechanisms and structure. These will be briefly introduced below, using a running example. In this section, we describe general CEP concepts and mechanisms, but to avoid ambiguity and for clarity and illustration purposes we will use the Esper EPL language.

1) RUNNING EXAMPLE. FIRE DETECTION IN A SMART HOUSE

To illustrate our proposal, suppose a neighborhood with smart houses, each one equipped with sensors to detect temperature and carbon monoxide (CO) levels. In particular, ceiling sensors record the absolute value of the temperature of a house, and CO detectors analyze the amount of CO gas present in the air. Additionally, a sensor detects whether the entrance door is open or closed. People living in that neighborhood are equipped with wristbands that permit knowing their location. Using the measurements provided by all these sensors, we are interested in monitoring the following situations:

- **TempIncrease**: The temperature of the house has increased 2 or more degrees in less than one minute.
- **TempWarning**: Four `TempIncrease` events, whose temperature is always above 33 degrees, occur in less than 5 minutes.
- **COHigh**: The CO level of a house exceeds 5000 units.
- **FireWarning**: A `COHigh` event is detected, followed by a `TempWarning` event, everything within less than 5 seconds.
- **NobodyHome**: The main door of the house is closed and there is nobody within the perimeter of the house.
- **CallFireDept**: A `FireWarning` event occurs after a `NobodyHome` event is detected. Therefore, the Fire Department should be called.

2) CEP EVENTS

Luckham [14] defines an event as a record of an activity that happens, or is contemplated as happening in a system. Every event has a type and a set of attributes. In most of CEP systems, events occur instantaneously, and they all have

a timestamp attribute (either implicit or explicit) indicating the moment in time when they are triggered.

For example, the following tuples are examples of `Person` and `Home` events. The system receives information periodically from the sensors and notifies it by means of `Home` events, which include information about the house `id`, the time at which the event was produced (`ts`), the coordinates (`x`, `y`) of the house, its size in square meters (`sq`), the temperature (`temp`) and carbon monoxide levels (`co`), and whether the door is open or not (`dopen`). Similarly, sensors in the wristbands of the people in the system emit information about them. Each `Person` event indicates the person `id`, the event timestamp (`ts`), and the coordinates of the person location (`x`, `y`). Times are expressed using the POSIX time convention, i.e., by the number of seconds elapsed since January 1, 1970 [31].

```
Person(id:1, ts:1533048980, x:50,  y:50)
Person(id:2, ts:1533048980, x:100, y:100)
Person(id:3, ts:1533048981, x:150, y:150)
Home(id:1, ts:1533048980, x:0, y:0,
     sq:100, temp:20, co:4000, dopen:false)
Home(id:2, ts:1533048980, x:0, y:200,
     sq:120, temp:20, co:4000, dopen:false)
Home(id:3, ts:1533048982, x:200, y:0,
     sq:150,temp:30, co:4000, dopen:false)
```

3) CEP RULE SPECIFICATION

A CEP rule defines a complex event, by means of a pattern expression that combines other (simple or complex) events. Whenever the pattern is detected in the stream (i.e., it is *satisfied* by the stream events), the complex event is created and added to the same stream. In the rest of this section, we will identify and describe the most basic and representative types of CEP patterns.

a: SELECTION PATTERNS

The simplest rule permits creating a complex event every time a given simple event is detected. For example, the following rule, named `COHigh`, creates a `COHigh` event every time a `Home` event is detected whose CO level exceeds 5000 units.

```
@Name('COHigh')
insert into COHigh
select h1.ts as ts, h1.id as id
from pattern[(every(h1=Home(h1.co>=5000)))];
```

In the pattern, the label `h1` acts as an alias that refers to an expression (in this case, the `Home` event) and can be used in other sub-expressions. The event `COHigh` has two attributes (`ts` and `id`) whose values are obtained from those of the `Home` event that triggers the rule. The use of the operator `every` ensures that a `COHigh` event is created every time a `Home` event satisfies the pattern. Otherwise, only one complex event will be created the first time a simple event satisfies the pattern.

b: TEMPORAL SEQUENCING OF EVENTS

One important CEP operator captures is the *sequence* that requires that all events occur sequentially. In Esper, this operator is called *followedBy* (“`->`”) and introduces a temporal ordering between pairs of events or expressions. Events related by this operator do not need to be consecutive: “`A -> B`” only implies that `A` occurs some time before `B`.

For example, the pattern `TempIncrease` is triggered whenever two `Home` events related to the same house occur within a window of 100 seconds, and the difference between the temperatures of the two is greater than 2.

```
@Name("TempIncrease")
insert into TempIncrease
select h1.id as id,
       h2.ts as ts,
       h2.temp as temp,
       h2.temp-h1.temp as incr
from pattern [every(h1=Home() -> h2=Home(
       h2.id=h1.id and h2.temp-h1.temp>=2))]
       .win:within(100 seconds);
```

c: WINDOWS

We can also assign *windows* to rules to restrict their scope. Windows could refer to specific time intervals (time windows) or the number of occurrences of events (event windows). Furthermore, in Esper each window can be either batch or sliding. *Batch windows* have fixed starting and ending points. In contrast, *Sliding windows* update its starting and ending points adding a unit to them continuously. In other CEP languages, such as the Azure Stream Analytics Query Language, batch windows are called *tumbling* windows and correspond to fixed-sized, non-overlapping and contiguous time intervals. *Hopping* windows are similar, although they model scheduled overlapping windows (in Esper, they can be defined using *contexts*). Finally, *session* windows group events arriving at similar times, but filtering out periods of time with no events. The examples in this paper use batch (i.e., tumbling) and sliding windows.

For instance, the previous rule `TempIncrease` used a batch time window of size 100, which means that the rule is triggered every 100 seconds. We can also define a *sliding time window* whose ending time is `T`, where `T` is the timestamp of the event being considered, and its starting time is $\mathtt{T} - \mathtt{L}$, with `L` the duration of the window. The rule `NobodyHome` described below uses a sliding time window of 3 seconds.

```
@Name("NobodyHome")
insert into NobodyHome
select p.ts as ts, h.x as x, h.y as y,
       h.id as id
from pattern [(every h =Home(not dopen) ->
  every (p=Person(
    (p.x <= (h.x - Math.sqrt(h.sq)/2)) or
    (p.x >= (h.x + Math.sqrt(h.sq)/2)) or
    (p.y <= (h.y - Math.sqrt(h.sq)/2)) or
    (p.y >= (h.y + Math.sqrt(h.sq)/2))) ))
where timer:within(3 seconds)];
```

Similarly, *event windows* permit referring to sets of particular events of a given size (the *window size*). For instance, every 100 `COHigh` events. Event windows can be either *batch* or *sliding*, too.

d: TIME MODEL

In general, the behavior of time windows, and of the *followedBy* operator ($->$), depends on the notion of time used by the CEP engine. Different CEP systems implement distinct time semantics, which can be classified into event time, ingestion time or processing time semantics [32], [33]. In *event time semantics*, each event is timestamped by the source that produces it. Under this model, events are always processed in the order they were generated, no matter the order in which they are received by the CEP engine, or the time at which they arrive. This model ensures determinism (as long as the CEP system is not probabilistic [22]).

In *ingestion time semantics*, the timestamp is assigned by the engine that receives the events (assuming a single machine receives and timestamps all events) and the original timestamps of the events are ignored. This model also ensures deterministic processing, but it does not respect the order in which the events were produced—they may arrive out of order due to communication or network delays, for example. Differences in the source clocks are resolved in this way.

Finally, in *processing time semantics*, the timestamp of the events are determined by the clock time of the physical machine processing them, at the moment in time when they are processed. Any source timestamp is ignored. Even when this is not a deterministic model, it is common in distributed processing systems under the assumption that the clock skew between physical machines processing the events is negligible, and that the CEP patterns processing time does not introduce significant delays [33].

To deal with uncertain timestamps, we need to either assume event time semantics in order to handle their values, or be able to get the timestamps from the processing engine (assuming they have uncertain values). In the following, to reason about the uncertainty of the timestamps we will use the values of the event attributes, no matter how they are obtained—i.e., we will use event time semantics.

e: COMBINATION OPERATORS

Expressions in rules can be combined in different ways by using logical operators (`or`, `and`, etc.), comparison operators ($<$, $\leq$, etc.), and temporal connectors (`until`, `while`, etc.), among others. In addition, windows can be combined restricting their scope.

We have used these kinds of operators in the `TempIncrease` and `NobodyHome` rules above. They are also used in the `TempWarning` rule, which starting from an event with temperature equal to or greater than 33 degrees, is triggered in case of increasing temperature of four `TempIncrease` events collected in a five minutes interval:

```
@Name ("TempWarning")
insert into TempWarning
select t1.id as id,
       t4.ts as ts,
       t4.temp as temp
from pattern [(every
 (t1=TempIncrease(t1.temp>=33))
 ->(t2=TempIncrease(t2.temp>t1.temp and
                    t2.id=t1.id))
 ->(t3=TempIncrease(t3.temp>t2.temp and
                    t3.id=t1.id))
 ->(t4=TempIncrease(t4.temp>t3.temp and
                    t4.id=t1.id)))
where timer:within(5 minutes)];
```

A similar rule, `FireWarning`, imposes a pattern that requires that a `TempWarning` event follows a `COHigh` event within less than 5 seconds.

```
@Name("FireWarning")
insert into FireWarning
select tw.id as id, coh.ts as ts
from pattern [(every (coh=COHigh())
    ->every (tw=TempWarning(tw.id=coh.id)))
where timer:within(5 seconds)];
```

Another complex event in our system is created by the rule `CallFireDept`, which generates an event of the same name when the system detects that there is nobody home and a `FireWarning` event is detected in the stream within 5 seconds:

```
@Name("CallFireDept")
insert into CallFireDept
select fw.id as id, fw.ts as ts
from pattern [(every (nh = NobodyHome())
    ->(fw = FireWarning(fw.id = nh.id)))
where timer:within(5 seconds)];
```

f: HIERARCHIES OF EVENTS

As mentioned above, CEP systems enable that complex events generated by a rule are used by other rules, thus generating hierarchies of rules and events [28]. Fig. 1 shows the hierarchy for the smart house case study, represented by means of a dependency graph. It shows the two simple events at the top (with dashed lines) and the CEP rules (with normal lines) by means of nodes. An arrow between two nodes A and B indicates that rule B depends on the rule (or simple event) A. In general, this graph should be acyclic [34].

In our case study, there are rules such as `TempWarning`, `FireWarning`, and `CallFireDept` that use other complex events in their specifications. These rules should take into account execution priorities or any other mechanism to guarantee their confluence and avoid event races [34], [35].

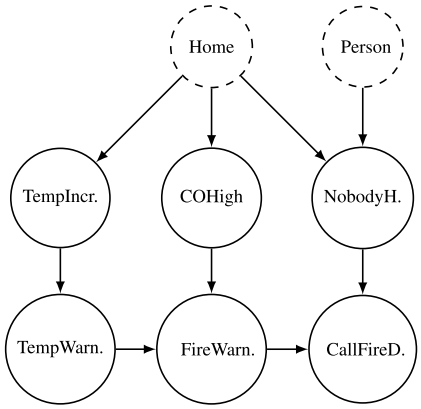

FIGURE 1. Rule dependency graph for the smart house case study.

4) STRUCTURE OF CEP RULES

An important feature of EPLs that we exploit in this paper is that their rules usually share a common structure:

- A **Selection** phase that identifies the occurrence of the source events that will trigger the pattern. These events constitute the *antecedents* of the complex event produced by the rule.
- A **Matching** phase that decides whether values of the attributes of the selected events fulfill the pattern requirements, and evaluates the combination of all pattern conditions (using operators `and`, `or`, $->$, etc).
- A **Production** phase that generates the complex event and calculates the values of its attributes. This step may also include the computation of aggregated values using sums, averages, and similar operations.

B. UNCERTAINTY

Uncertainty is the quality or state that involves imperfect and/or unknown information. It applies to predictions of future events, estimations, physical measurements, or unknown properties of a system [36]. For example, measurement uncertainty refers to the inability to know with complete precision the value of a quantity. Confidence refers to the degree of belief that we have on the actual existence of an entity (in our case, the occurrence of an event), or on the inference of a given CEP rule.

Both types of uncertainties can be represented in different forms, and using different formalisms. For example, measurement uncertainty can be expressed by means of a probability distribution associated to every uncertain variable, representing the distribution of the dispersion of its values. This is the approach used by, e.g., the UML Profile for MARTE [37]. However, this approach represents some limitations when calculating the aggregated uncertainty of the result of an operation that involves operands with different probability distributions. A more widely adopted approach among engineers of different disciplines to represent measurement uncertainty, is defined by the Guide to the Expression of Uncertainty in Measurement (GUM) [38]. This international standard defines general rules for evaluating and expressing uncertainty in measurement of physical quantities. It provides a quantitative indication about the quality of the result so that those using it can assess its reliability. The GUM associates a *standard uncertainty* to any uncertain value, defined by the standard deviation of the measurements for such a value. Then, a Real number x becomes a pair (x, u), also noted $x \pm u$, that represents a random variable X whose average is x and its standard deviation is u. With this, if X follows a normal distribution $N(x, u)$, we know that 68.3% of the values of X will be in the interval $[x - u, x + u]$.

Uncertainty also applies to Boolean values. For example, in order to implement equality and comparison of numerical values with uncertainty, the traditional values of `true` and `false` returned by Boolean operators are not enough. Comparisons must to return Real numbers in the range [0, 1], representing the probabilities that one uncertain value is equal, less or greater than other. For instance, if $a = 2.0 \pm 0.3$ and $b = 2.5 \pm 0.25$, then we have that the probabilities of $a < b$, $a = b$ and $a > b$ are 0.8937, 0.1062, and 0.0001, respectively. This leads to the definition of *Uncertain Booleans*, which are Boolean values accompanied by the level of confidence that we assign to them [23]. Therefore, a `UBoolean` value is a pair (b, c) where b is `true` or `false`, and the number $c \in [0, 1]$ represents the probability that b is correct. Thus, possible uncertain Boolean values are $(true, 0.95)$ or $(false, 0.8)$. It also complies with properties such as $(b, c) = (\neg b, 1 - c)$. `UBoolean` is a proper supertype of `Boolean` and its associated operations, where *true* corresponds to $(true, 1.0)$ and *false* to $(true, 0.0)$. The logical operators, such as `and`, `or`, `not` or `implies`, are also lifted to the `UBoolean` supertype and therefore extended to deal with confidence. For example, assuming that two logical variables $(true, c_1)$ and $(true, c_2)$ are independent, then $(true, c_1) \wedge (true, c_2) = (true, c_1 * c_2)$ [23].

To deal with uncertainty, we have used the *U-Model* conceptual model [39] as a reference framework. When it comes to concrete concepts that specialize some of the high-level U-Model concepts, such as measurement uncertainty, or confidence, we use standard references. In particular, we rely on the ISO International Vocabulary of Metrology (VIM) [40] and the GUM [36] for all measurement uncertainty related matters, and we use probability theory [41] to express *Confidence* and to assign probabilities to both events and CEP rules. The confidence that we give to an event represents the degree of trust that we have on their actual existence.

Here, it is interesting to distinguish between the *real* and the *modeled* events: the former ones happen in reality; the latter are the ones included in the stream of events and processed by the CEP system, representing the real ones. It may be the case that the event exists in reality but the CEP system has not captured it (i.e., a false negative), or that the CEP rules generate an event that does not actually exist (i.e., a false positive). The first case may be due to unreliable sources (e.g., defective sensors) or unreliable network that may drop packages, while the second case is normally due

to duplication of packages at network level, or to unreliable CEP rule that erroneously produce complex events when they should not.

We use probability theory, and not *fuzzy logic* because, although both approaches deal with states of uncertainty, and represent degrees of subjective belief, fuzzy set theory uses the concept of fuzzy set membership, whereby an element can belong to different sets of one partition of the whole space. In turn, probability theory uses the concept of subjective probability, i.e., the likelihood of an event or condition to belong to each set of the partition, assuming it can only belong to one set [42].

In a previous work [23], we showed how measurement uncertainty can be incorporated into OCL [24] and UML [25] primitive datatypes, including `Real`, `Integer` and `Boolean`, defining super-types for them: `UReal`, `UInteger` and `UBoolean`, respectively. We also implemented in Java all the operations on these extended types to allow modelers to use them and to propagate the uncertainty through the operations.

The incorporation of such a library for extended datatypes into CEP systems enables users to define and manipulate uncertainty in the CEP rules in a high-level and transparent manner. In this way, users do not need to worry about the propagation of uncertainty, which is automatically performed by the operations of these types.

C. UNCERTAINTY IN CEP SYSTEMS

Based on the concepts defined above, we have identified different kinds of uncertainty that may happen in CEP systems. Let us describe them using the common structure of the CEP rules. Starting with the selection phase:

- Uncertain events in the stream: missing events in the stream, despite the fact that they actually happened (we refer to them as false negatives, FN); or events in the stream that were wrongly inserted (false positives, FP).
- Lack of precision in the values of the attributes of the basic events in the stream, due to imprecision of the measuring methods or tools (measurement uncertainty). This kind of uncertainty includes lack of precision in the events timestamps [43].

In the matching phase:

- Lack of precision due to uncertainty of comparison operators ($=$, $<$, $>$,...) between uncertain values of attributes of matched events. For example, when comparing two real values with uncertainty, such as $a = 2.0 \pm 0.3$ and $b = 2.5 \pm 0.25$, we obtain that $a < b$ with a confidence of 0.893 [23]. Any decision that we make based on such a comparison should then be subject to uncertainty. Note that this case also includes the $->$ operator, since it basically compares the timestamps of the events.
- Lack of precision due to uncertainty of logical composition operators (`or`, `and`, `not`) between uncertain statements. Note that under the presence of uncertainty, these operators no longer return a Boolean value but a `UBoolean` that represents the confidence that we assign to the result of the operation.

In the production phase:

- Lack of precision in the values of the attributes of complex events, due to the propagation of measurement uncertainty in their computation from the events' attributes.
- Lack of precision in the probability of the occurrence of the generated complex event, due to incomplete or erroneous assumptions about the environment in which the system operates, which may influence the confidence we have in the CEP rule.

The following sections are dedicated to discuss the existing approaches for dealing with these kinds of uncertainties, and how we propose to address them. Basically, we focus on two kinds of uncertainties: measurement uncertainty (due to imprecision in the possible values of the attributes of events) and occurrence uncertainty (the confidence that we assign to the events in the streams, to the ones produced by the rules, and to the rules themselves). Measurement uncertainty is a particular type of *aleatory* uncertainty, whilst occurrence uncertainty is a type of *epistemic* uncertainty [44].

III. EXTENDING CEP WITH MEASUREMENT UNCERTAINTY

This section describes how we have incorporated uncertainty information to CEP systems. More precisely, we discuss how measurement uncertainty is associated to: (*a*) events, both simple and complex—i.e., the probability that can be assigned to their occurrence; (*b*) the values of their attributes, since they may be produced by imprecise or intermittently faulty sensors (in case of simple events), or derived from uncertain data (in case of complex events); (*c*) their timestamps, because clocks may not be accurate or not synchronized; and (*d*) the CEP rules, since sometimes we are not completely confident about their inferences.

A. UNCERTAINTY IN MEASUREMENTS AND ATTRIBUTES VALUES

In order to include measurement uncertainty in the attributes of events, we declare the attributes using the corresponding uncertain datatype (`UReal`, `UInteger`, `UBoolean`, etc.). By using our library [23], users do not need to worry about the propagation of uncertainty, since it is performed internally and in a transparent way by the corresponding operations of each extended type. Although in many cases the events are independent, our proposal also permits dealing with dependent events, as discussed later in Sect. III-F.

B. ASSIGNING UNCERTAINTY TO TIMESTAMPS

In our proposal we assume an *event time semantics* time model [33], whereby events are explicitly timestamped by their sources, and therefore we assume that events have an attribute that contains the value of such a timestamp.

Similarly to other proposals, e.g., in [20], we consider that timestamps may also have some associated measurement uncertainty. Therefore, to represent timestamps we use either the datatype `UInteger` or `UReal`, depending on whether time is discrete or continuous in our system. These extended types incorporate a real number to the base value, which indicates its precision, in terms of the standard deviation of the possible values [23], [36]. For example, if our timestamps are expressed in milliseconds using the POSIX notation [31], a possible value for a timestamp produced by a clock whose precision is 1 ms, can be 1544616725034 ± 0.58.[4] Notice how the comparison between two timestamps returns a `UBoolean` value, since now it is difficult to tell apart two timestamps that differ in less than 1 ms. In fact, the comparison $34 \pm 0.58 < 35 \pm 0.58$ yields a confidence of just 0.62. However 34 ± 0.58 is less that 36 ± 0.58 with a confidence of 0.92, and less than 37 ± 0.58 with a confidence of 0.99. Since the uncertainty is associated to the individual values of the timestamp attributes, we allow the value of the uncertainty to vary from one clock to another, and even vary among different timestamps of the same clock—hence enabling the simulation of the degradation of the clocks' precision when required.

C. ASSIGNING PROBABILITIES TO EVENTS

We also need to assign probabilities to events, expressing the *confidence* that we have on their occurrence. In this proposal, we do this by adding an attribute called `conf` to every event.

This confidence represented as a probability does not mean how often the event is supposed to happen—something that for simple events can be expressed in terms of a probability distribution, or for complex events can be estimated using, e.g., Monte-Carlo simulations [35]—, but the confidence level that we have on the fact that the event actually occurs in reality if the CEP system predicts its occurrence.

For a simple event `e`, this probability coincides with $(1 - P_{fp}(\mathtt{e}))$, where $P_{fp}(\mathtt{e})$ is the probability of a false positive for that event. This information is normally obtained from the sensor manufacturer, or similar sources. It may also be due to unreliable communication networks that duplicate packets, or other causes. In any case, our contribution is a mechanism for representing and taking into account this information explicitly in our events.

In our proposal we are able to deal with false negatives too. Although these situations are rare in practice, there are cases in which the critically of missing a particular kind of event cannot be neglected. Sect. III-G describes how we deal with false negatives.

[4] According to the GUM [36], assuming a uniform distribution of the possible values of the variable, the value is taken as the midpoint of the interval, $x = (a+b)/2$, and its associated uncertainty as $u = (b-a)/(2\sqrt{3})$. In this case, $b - a = 2$ and therefore $u = 0.58$.

D. ASSIGNING PROBABILITIES TO COMPLEX EVENTS

Since complex events are derived, we have to derive also their probabilities. Our approach assumes that the probability of a complex event in CEP depends on three main aspects: the confidence that we have on the occurrence of the input events of the pattern (*antecedents*); the confidence level that we have on matching and comparison operations performed by the rule pattern to trigger the production of the complex event, as well as the computation of its attributes; and, finally, the confidence that we have on the rule itself.

This means that, given a rule R whose antecedents are events $\mathtt{e}_1, \ldots, \mathtt{e}_n$ (they can be either simple or complex events), that performs a matching process m_R and produces a complex event `e`, the probability of complex event `e` is given by the following expression

$$P(\mathtt{e}) = P(\mathtt{e}_1, \ldots, \mathtt{e}_n) \cdot P(m_R) \cdot P(R) \tag{1}$$

where:

- $P(\mathtt{e}_1, \ldots, \mathtt{e}_n)$ is the combined probability of the events. For example, in case they are all independent, $P(\mathtt{e}_1, \ldots, \mathtt{e}_n) = P(\mathtt{e}_1) \cdot \ldots \cdot P(\mathtt{e}_n)$. Otherwise conditional probabilities should be used, as detailed later in Section III-F.
- $P(m_R)$ is the confidence level of the matching process, which not only accounts for the uncertainty in the comparison operations between uncertain values and their combination using logical connectors (`or`, `and`, $->$, etc.), but also the propagation of uncertainty in the operations that calculate the values of the attributes of the complex event.
- $P(R)$ is the rule confidence, represented by a probability that captures the possible imprecision due to incomplete or erroneous assumptions about the environment in which the system operates, or the knowledge we have about the system behavior. This confidence can be calculated by Bayesian Networks, as in [21], by expert knowledge, or by any other means.

As we can see, the inputs of this method are the confidence of the simple events (i.e., the probabilities of being false positives and, if required, false negatives); the conditional probabilities of the dependent events; and the probability of the rule, as estimated by expert users.

E. THE SMART HOUSE EXAMPLE WITH UNCERTAINTY.

Let us show how the probabilities of the simple and complex events for the smart house example are calculated. Starting from the simple events, they are enriched with the measurement uncertainty of their attributes, and with their confidence. Remember that, for simple events, such as confidence is given by the probability of a false positive for that event, and stored in the event's `conf` attribute.

The following two simple events are examples of those that will be fed into the CEP system:

```
Person (id:1, ts:(153304898,1.0),x:(50,0.1),
        y:(50,0.1), conf:0.999)
Home (id:3, ts:(153304898,1.0), x:(0,0.1),
      y:(0,0.1), sq:(100,0.1),
      temp:(20,0.3), co:(4000,20),
      dopen:(false,0.99), conf:0.998)
```

The probability of each complex event is calculated as the product of three factors: (1) the confidence level on its *antecedents*, (2) the confidence level on matching and comparison operations, and (3) the confidence level on the rule itself. E.g., the probability of the `TempIncrease` event, created by the rule of the same name, is given by:

$$\begin{aligned} &P(\texttt{TempIncrease}) \\ &= P(\texttt{Home})^2 * && \textit{//Antedecents} \\ &P\,(\texttt{h2.temp} - \texttt{h1.temp} \geq 2.0) * && \textit{//Matching ops.} \\ &P(\texttt{h1.ts} < \texttt{h2.ts}) * && \textit{//Comparison ops.} \\ &P\,(\textit{TempIncreaseRule}) && \textit{//Rule confidence} \end{aligned} \tag{2}$$

One of the benefits of using Esper EPL is that it permits the use of external libraries written in Java, whose methods can be invoked in the CEP rules. The only limitation is that these methods should be static. Then, in order to use our library of operations for the extended types [23], we have built wrapper classes for each of the types, with static methods. For example, class `UReals` contains the static implementations of all methods provided by class `UReal`. Similar classes have been developed for the rest of the uncertain datatypes.

With this, the rule `TempIncrease` can be written with uncertainty as follows.

```
@Name("TempIncrease")
insert into TempIncrease
select h2.ts as ts,
     h1.id as id,
     h2.temp as temp,
     UReals.minus(h2.temp,h1.temp) as incr,
     // Antecedents
     h1.conf * h1.conf *
     // Comparion operations
     UReals.ge(UReals.minus(h2.temp,
       h1.temp),2.0).getC() *
     UReals.lt(h1.ts,h2.ts).getC() *
     // Rule confidence
     P(TempIncreaseRule) as conf
from pattern [(every (h1 = HomeEvent() ->
     h2=HomeEvent(UBooleans.toBoolean(
         UReals.ge(UReals.minus(h2.temp,
         h1.temp),2.0)) and h2.id=h1.id)))
where timer:within(1 minutes)];
```

We can see how the rule computes all attributes of the complex event, including attribute `conf` with the associated confidence, using the formula (2) above. It uses the probability of the antecedents, the confidence of the operations (method `getC()` applied to a `UBoolean` value returns its confidence) and the probability of the rule (a function `P(TempIncreaseRule)` returns that value). Notice that we have separated the computation of the rule confidence from its application in the formula. In this way, the function `P(TempIncreaseRule)` could be a constant, a function or the output of a Markov Logic Network or a Bayesian Network defined for that rule [21].

The rest of the rules of our system follow the same strategy. The interested reader can consult them in our project web site and in our Git repository [45].

This example also illustrates the need to consider probability in CEP systems instead of all events being equally probable. This introduces an implicit prioritization mechanism, very useful for instance when two or more critical events occur (e.g., `CallFireDept`). In these cases, we could discriminate among them based on their probability, attending those most probable. These are those in which we have more confidence that they have been correctly inferred.

F. DEALING WITH DEPENDENT EVENTS

So far we have assumed that events are independent from each other, and that the values of its attributes are independent variables too, i.e., the confidence we have on the occurrence of one of them does not depend on the confidence we have on the occurrence of others.[5] Although this is the common assumption in most proposals [22], in this section we discuss how to deal with dependent events and dependent attributes.

1) CALCULATING THE CONFIDENCE OF DEPENDENT EVENTS

A common situation of dependence between the confidence of events happens when a complex event B depends on a complex event A, and both depend on a simple event e. In this case, the confidence of A already took into consideration the confidence of e, and therefore we cannot simply multiply the confidences of A and e to obtain the confidence of B, because we would be counting twice the confidence of e.

These cases always occur during the generation of complex events that depend on more than one complex event (i.e., when complex events have more than one incoming arrow from other complex events in the dependence graph). For example, in the smart house case study, the antecedents of the event `FireWarning` are the events `COHigh` and `TempWarning` (see Fig. 1). The event `TempWarning` also depends on event `TempIncrease`, which in turn depends on `Home`. The probability of the generated `FireWarning` event cannot be computed as the product of the individual probabilities, since the antecedent events are not independent. The situation is similar for complex event `CallFireDept`.

In these cases, the probability of the combined event, $P(\mathsf{e}_1, \ldots, \mathsf{e}_n)$, is $P(\mathsf{e}_1)\cdot P(\mathsf{e}_2|\mathsf{e}_1)\cdot P(\mathsf{e}_3|\mathsf{e}_1 \wedge \mathsf{e}_2)\cdot \ldots \cdot P(\mathsf{e}_n|\mathsf{e}_1 \wedge \ldots \wedge \mathsf{e}_{n-1})$, where $P(\mathsf{e}_i|\mathsf{e}_1 \wedge \ldots \wedge \mathsf{e}_{i-1})$ is the conditional

[5] It is important to note here, again, that our confidence does not express the probability of an event to happen, but the degree of belief that our CEP rules have correctly inferred it.

probability that event e_i is correctly inferred, given that events $e_1, \ldots, e_{i-1}$ have been correctly inferred.

To compute the conditional probability of events, we need to consider the number of incoming paths that the node representing the event has from a common ancestor in the rule dependency graph, and eliminate all the duplicated probabilities from that ancestor. For example, in the smart house case study (whose rule dependency graph is shown in Fig. 1), event `FireWarning` receives two incoming paths from a common ancestor: `Home-TempIncrease-TempWarning-FireWarning`, and `Home-COHigh-FireWarning`. Therefore, we should consider the confidence of event `Home` only once when computing of the confidence of event `FireWarning`. Similarly, there are three paths that lead to event `CallFireDept` from a common ancestor, `Home`, and therefore we should consider its confidence only once—and not three times, as we would be doing if we simply multiplied the probabilities of the antecedents of event `CallFireDept`.

With this, assuming that all the events are independent, the confidence of `FireWarning` events is computed as follows:

$$\begin{aligned} &P(\texttt{FireWarning}) \\ &= \texttt{tw.conf} * \texttt{coh.conf} * \quad \textit{// Antedecents} \\ &\texttt{coh.ts.uLt(tw.ts).getC()} * \textit{// Comp. operations} \\ &P\,(\textit{FireWarningRule}) \quad \textit{// Rule confidence} \end{aligned} \tag{3}$$

Thus, the rule `FireWarning` could be specified in Esper EPL as:

```
@Name("FireWarning")
insert into FireWarning
select tw.id as id, coh.ts as ts,
  // Antecedents
  tw.conf * coh.conf *
  // Comparison operations
  UReals.lt(coh.ts,tw.ts).getC()*
  // Rule confidence
  P(FireWarningRule) as conf
from pattern [(every(coh=COHigh()) ->
  every(tw = TempWarning(tw.id=coh.id)))
where timer:within(5 seconds)];
```

However, the events `FireWarning` and `COHigh` are not independent, since they both depend on the simple event `Home`, and therefore their probabilities should not be simply multiplied. Thus, the formula (3) needs to be reformulated and now the probability of the antecedents is no longer $\texttt{tw.conf} * \texttt{coh.conf}$ but $P(\texttt{tw}|\texttt{coh})$, which coincides with $\texttt{tw.conf} * \texttt{coh.conf/home.conf}$. Hence, the correct way to calculate the probability is:

$$\begin{aligned} &P(\texttt{FireWarning}) \\ &= (\texttt{tw.conf}*\texttt{home.conf})*\texttt{coh.conf}* \textit{// Antedecents} \\ &\texttt{coh.ts.uLt(tw.ts).getC()} * \quad \textit{// Comp. ops.} \\ &P\,(\textit{FireWarningRule}) \quad \textit{//Rule conf.} \end{aligned} \tag{4}$$

And the correct implementation of rule `FireWarning` in Esper EPL is:

```
@Name("FireWarning")
insert into FireWarning
select tw.id as id, coh.ts as ts,
       // Confidence of antecedents
       (tw.conf/Home.conf)*coh.conf*
       // Comparison operations
       UReals.lt(coh.ts,tw.ts).getC()*
       // Rule confidence
       P(FireWarningRule) as conf
from pattern [(every(coh=COHigh())
    -> every(tw=TempWarning(tw.id=coh.id)))
where timer:within(5 seconds)];
```

In our implementation, we have assumed that all `Home` events have the same confidence, and have stored it as a global attribute of a class `Home`. This is a common situation when the probability of a false positive of a given event is a value that depends on the sensor that produces this type of event, and therefore the same for all these events. If this were not the case in a particular application, it would be just a matter of adding attributes to the complex events, with the confidences of their relevant ancestors, so that they are available in the rules where they act as antecedents.

2) CALCULATING THE VALUES OF DEPENDENT VARIABLES

The attributes of the events, whose values are used in rule patterns to decide whether the rules should be triggered or not, or to compute the values of the attributes of the complex event generated by the rule, might not be independent. For example, the level of CO has a direct influence on the temperature of the house, and therefore an increase in CO normally ends up producing an increase in the temperature. Since both attributes are subject to uncertainty, we also need to consider their dependency when computing the uncertainty of the result of an operation that combines them.

The process of dealing with the uncertainty of dependent variables is described in the GUM [36], and requires taking into account not only the variance of the possible deviations of the values of the attributes, due to their measurement uncertainty, but also the covariance of their joint variations. In our case, this would affect the resulting uncertainty of those operations performed on dependent uncertain Reals, and the associated confidence that we obtain when comparing them. For this, our Java library supports an additional parameter in operations with UReals that can be used to provide the correlation coefficient of the two variables to combine.

G. DEALING WITH FALSE NEGATIVES

The concept of "false negative" may have different meanings depending on the context in which it is applied. In our case, this term denotes the non-detection of (simple or complex) events in the stream despite the fact that they have actually occurred. Sometimes the loss or non-detection of an event is not very relevant. However, missing the occurrence of one particular event in some critical applications may have

significant consequences—mainly if it is the antecedent of a rule that creates an essential complex event for the application. In these situations, it is important to consider the possibility of false negatives in the stream.

Some authors (e.g., [46]) use mining techniques to detect false negatives. Other proposals, mainly coming from the Very Large Databases (VLDB) community, use techniques for the efficient representation of all possible worlds, in order to query systems with incomplete information [47]. In our work, we follow this latter approach and define *synthetic rules* that capture the situation where an antecedent event is lost, and therefore it is missing in the rule.

Thus, given a rule with a pattern P and with antecedent events $\{e_1, \ldots, e_n\}$, a synthetic pattern $P_{\{e_{i1},\ldots,e_{ik}\}}$ is derived from P by removing antecedent events $\{e_{i1}, \ldots, e_{ik}\}$ and by adding a condition that these events are not currently present in the stream. This simulates the fact that the "removed" events have actually happened, but they were lost.

To illustrate how these synthetic rules are defined, we will use the Motorbike example later detailed in Sect. IV-A.4 because these kinds of false negatives events match better with the kinds of events handled in that case study. In particular, the Motorbike application defines one rule that creates a `DriverLeftSeat` event when the seat sensor detects that the driver is no longer sitting on the motorbike. The events generated by this rule are used to detect accidents when the motorbike is moving, the speed suddenly decreases, and the driver is thrown out from the seat. In a real accident, however, chances are that a simple event may be lost, or it is not inserted in the stream, and hence the need to consider this possibility in the production of critical complex events, such as the one detecting an accident.

The probabilities of the false negatives need to be provided by the system specifier. In this case, we assumed that this probability is a constant, defined by attribute `confFN` in class `Motorbike`. With this, the synthetic rule that represents the loss of event `Motorbike` in rule `DriverLeftSeat` could be written as follows.

```
@Name('DriverLeftSeat')
@Priority(2)
insert into DriverLeftSeat
select current_timestamp() as timestamp,
    e2.motorbikeId as motorbikeId,
    e2.location as location,
    true as seat_a1,
    e2.seat as seat_a2,
    e2.conf*Motorbike.confFN*
     P(DriverLeftSeat) as conf
from MotorbikeEvent as e2
where e2.seat = false and
not exists (select motorbikeId
from MotorbikeEvent
   .win:time(5 milliseconds) as e1
where e1.motorbikeId=e2.motorbikeId);
```

In general, some considerations should be taken into account when defining synthetic rules. First, they only make sense for rules that deal with simple events, since complex events are generated by the CEP system and therefore there is no point in thinking that they may be lost. Furthermore, complex events already incorporate their confidence in the form of probabilities, thus they already capture the fact that they might not be representing the actual events properly. Second, they normally make more sense in the case of sporadic (i.e., non-periodic) events, whose loss is more difficult to detect. From a technical perspective, they should have less execution priority than normal rules, since synthetic rules should only be triggered when the normal rules have not been triggered. Finally, it is the domain expert who should decide in which situation and for which events a synthetic rule must be defined, given that they introduce more complexity into the system. Incorporating these kinds of synthetic rules to the rule set of the CEP application may have a some impact on the number of produced events, and therefore on the performance of the system—we analyze this later in Sect. IV-C.2. Thus, it is up to the CEP system developer to decide when to use these synthetic rules, depending on how critical the loss of a particular kind of event is, and how likely this situation is. In the next section, we explain how to combine these synthetic rules with the use of thresholds to mitigate these problems.

H. THRESHOLDS IN RULES

In general, the probabilities associated to the complex events generated by the synthetic rules described above will be very low, and hence there might be no real need to produce them. Otherwise we may be populating the stream with events whose probability to occur is almost nil.

For instance, we can decide to create a synthetic rule only if the probability of the false negative of the event (i.e., the probability of missing it, despite having occurred) multiplied by the probability of the rule (that indicates our confidence on it) is above a certain threshold, e.g., 0.5. This is an a priori decision, which may help deciding whether the rule is created in the first place, or not.

For those rules defined for the CEP system, it is also possible to define thresholds that may help preventing the production of events with very low confidence. These thresholds can be defined both for the antecedents and for the generated events, preventing their production if the confidence of the events is below these thresholds. For example, a rule can state in its pattern a minimum threshold for the confidence of one or more of their ancestors, or for their combined confidence. If such a confidence does not reach the threshold, the rule is not triggered. Similarly, a CEP rule can calculate the confidence of the complex event it generates, and use this value in the rule pattern to decide whether it is worth triggering the rule or not.

Section IV-D discusses the effects of the use of thresholds in rules, regarding the accuracy of the resulting application.

IV. EVALUATION

This section describes the evaluation we have conducted to assess our proposal according to its correctness, performance,

accuracy, and applicability. This evaluation process follows the one defined in [48], which in turn is inspired in Adzic's adaptation of Moslow pyramid of human needs to software quality [49]. Each criteria provides the necessary foundations for the higher-level criteria. In the following, we discuss the extent to which we can claim our proposal meets these criteria. We finalize by identifying some of the current limitations of our proposal, and with some lessons we have learned while developing and evaluating our work.

A. APPLICATIONS

We have applied our approach to implement five different CEP applications that are subject to uncertainty. The applications have been chosen because they exhibit different characteristics, and come from different domains. They are either examples used in the CEP literature (Smart House [45], Tunnel Ventilation System [21], Motorbike [34]) or taken from real applications (the Madrid 2018 Marathon and the Andalusian Air Quality Monitoring application).

The applications have been implemented first in plain Esper, i.e., using basic events without uncertainty. Afterwards, these implementations (and their corresponding events) have been extended to account for measurement uncertainty in the values of their attributes and rules.

With these experiments, our goal was to study whether we are able to express and manage the different kinds of uncertainties that happen in these systems, as well as the overhead introduced by the uncertainty and the impact that the number of events to process, the number of rules, and the complexity of the rules patterns introduce—in terms of the number of arithmetic, comparison and logical operations. To check the applicability of our proposal to different CEP languages and engines, the experiments were also conducted using Apache Flink, obtaining very similar conclusions. These results are reported in Sect. IV-E

The following subsections present the case studies we have selected to evaluate our proposal with a description of their main characteristics in increasing order of complexity. The rules of all CEP applications (with and without uncertainty), the projects to execute them, and the different sets of input events used in the tests, can be downloaded from our web site and from our Git repository [45].

1) THE SMART HOUSE CASE STUDY

The Smart House case study has already been described in this paper, it has been used to illustrate the CEP concepts and also our proposal. It defines two kinds of simple events (`Home` and `Person`) and six rules, organized in a hierarchy tree of depth four and width three.

2) THE TUNNEL VENTILATION SYSTEM

This case study was originally presented by Cugola *et al.* in [21] to introduce CEP2U. It represents a tunnel ventilation system (TVS) constantly monitored by several sensors to detect possible failures, such as obstructions. In usual setups, sensors are evenly distributed along the tunnel in sectors (each one every 200m, for instance), and they measure the temperature (`temp`), the concentration of oxygen (`O2`) in air, and the presence of traffic jams. A CEP application designed to detect TVS malfunctioning has to recognize critical situations from the raw data measured by the tunnel sensors. Depending on the environment, the application requirements, and the user preferences, the presence of a TVS malfunctioning can be due to several reasons, each one captured by a separate CEP rule that can generate a `TVSMalfunctioning` event:

R1. The oxygen concentration in a tunnel sector is lower than 18%, and the temperature in the sector has gone above 30 degrees at least once during the previous 5 min.
R2. The temperature of a tunnel sector is higher than 30 degrees in absence of a traffic jam in that sector.
R3. The oxygen concentration of a tunnel sector is lower than 18% and the average temperature in the last 5 min is higher than 30 degrees.

3) AIR QUALITY MONITORING SYSTEM

This real case study corresponds to the CEP application that monitors the air sensor network for controlling the air quality of the Andalusian region of Spain. This network is composed of several sensor stations that gather the air pollutant measurements and submit them to a main server. Each station has sensors in charge of measuring six air pollutants: PM2.5, PM10, CO, O3, NO2 and SO2—see [34] for a complete description of this application. Seven CEP rules for each pollutant generate complex events that warn about the air quality level, in a 7-value scale, from `Average` to `Hazardous`.

The hierarchy tree is very different in this case: it has just a depth of 2, with a simple event at the top, and the 43 CEP rules in level 2. An example of a simple event that reaches the server (without uncertainty) is as follows:

```
Station(ts:1480546800, id:3,
    pm2_5:-1.0, pm10:-1.0, o3:0.043,
    no2:10735456, so2:39346848, co:0.302)
```

4) A FLEET OF MOTORBIKES

Suppose a fleet of motorbikes equipped with sensors that produce real-time information about their state, such as the pressure of their two tires, their location and speed. They also sense whether the driver is on the seat or not. The system goal is to be able to detect flat tires and vehicle crashes in real time, as well as to identify dangerous locations. These are the complex events that the system generates:

- **BlowOutTire**: The pressure of one of the tires of a moving motorbike goes down from more than 2.0 BAR to less than 1.2 BAR in less than 5 seconds.
- **Crash**: The speed of a motorbike goes from more than 50 km/h to 0 km/h in less than 3 seconds.
- **DriverLeftSeat**: The seat sensor detects that the motorbike driver has left the seat.

- **Accident**: A moving motorbike suffers a blow out of one of its tires, then a Crash event is detected, and the driver is thrown out, everything within less than 3 seconds.
- **AccidentsReport**: A complex event with the number of Accidents per day and location.
- **DangerousLocation**: This event is raised every time 100 events of type Crash are detected in a given location.

This example was used in [34] to analyze the confluence of the rules of CEP applications. An example of a simple event (without uncertainty) received by the application is shown below:

```
Motorbike(ts:1488326401, id:1,
  location:"Cadiz", speed:90.3,
  pressure1:3.10, pressure2:3.09, seat:true)
```

5) THE MADRID 2018 MARATHON

This application analyzes the information obtained from tracking the runners of the Madrid 2018 Marathon. Each runner was equipped with a sensor that generated events every time the runner passed one of the control points established every 5 km (plus one at km 21.0975, the half of the marathon, and one at the finishing line, km 42.195). The CEP rules are in charge of detecting the following situations, which should generate the corresponding complex events:

- **Escaped**: When the passing time at one of the control points between a runner and the next one exceeds 30 seconds.
- **TheWall**: The passing time of a runner between Km 35 and Km 30 is greater than the passing time between Km 30 and 25, plus 1 minute.
- **NegativeSplit**: The time taken by a runner in the second half marathon is less than the time taken to complete the first one.
- **Optimal**: A runner finishes the race with a negative split and without suffering the wall.
- **Cheated**: When the runner finishes the marathon but has missed the control point at Km 21.097 (half marathon).
- **Record**: A runner finishes the race in less than 2 hours, 3 minutes and 38 seconds.

An example of one of the events managed by the application (extracted from the data provided by the organization) follows below:

```
Runner(point:10, position:5520, dorsal:104,
  name:"Boris", surname:"Shvalev",
  category:"D-M", officialT:"4:20:58",
  netT:"4:20:50", netTS:15650)
```

B. CORRECTNESS

This section describes how we have evaluated the correctness of our proposal to represent and deal with the uncertainty of the occurrence of the events, the values of their attributes, their timestamps, and the confidence that we have on the rules. For this, we have conducted several tests, each one addressing a different correctness aspect.

In the first place, we checked that each CEP application extended with uncertainty behaves exactly the same as the application developed in plain CEP (i.e., without uncertainty) when the uncertainty was *nil*, i.e., when the confidence of all basic events is 1.0, the measurement uncertainty of the values of all attributes is 0.0, and the confidence of all rules was 1.0. In order to do this, we executed all applications with input sequences of events of size 10K, 100K, 200K, 300K and 400K events, with no uncertainty. Afterwards, we added *nil* uncertainty to the same sequences of events as described above and executed all the CEP applications again. The results confirmed that the sequences of generated events without uncertainty and with nil uncertainty where the same, i.e., contained the same events in the same order.

In order to test our system under real uncertainty (i.e., non-nil uncertainty) and using the same sequences of events, we added measurement uncertainty to the attributes of the events, and confidence to the events and rules, and executed all applications again. We used the same datasets as before with the only difference that the uncertainty associated to the event attributes and confidence was not nil anymore, but very small. Again, we observed that the output of the CEP system after its execution with and without uncertainty was the same. Of course, increasing the uncertainty of the attribute values may make the system become non-deterministic [17]. This will be discussed later in Section IV-D, when we discuss about the accuracy of our approach.

Another aspect to treat when talking about the correctness of our approach concerns the correctness of the propagated uncertainty in the generated event attributes and confidence. In this case, we rely on the correctness of the library, which was tested as part of our previous work [23].

C. PERFORMANCE

Performance is one of the common problems of all the proposals that incorporate uncertainty into CEP systems, given the potential degradation of performance [22] due to the operations required to compute the aggregated uncertainties and the probabilities of the complex events. The lack of standard benchmarks for CEP systems is a well known issue too, which hinders the performance analysis, and the comparison between different proposals [17], [21]. In this paper, we focus on the evaluation of the performance overhead introduced by the use of uncertainty in the events, in their attributes, in their timestamps, and in the rules (Sect. IV-C.1). We will assess in Sect. IV-C.2 the performance penalty introduced by the use of synthetic rules to deal with critical false negatives.

1) OVERALL PERFORMANCE OVERHEAD

For each of the five case studies, one table and two figures are shown, summarizing the results obtained when running the experiment for input event sets of sizes 10K, 100K, 200K, 300K and 400K. The table shows, for every set of input events, the time taken by the Esper CEP engine to process them (with and without uncertainty), the number of rules that were triggered (and hence the number of generated

		Performance				
	#Events	Time (s)	Ovhd.	#T.Rules	#Opers.	Ovhd.
Plain CEP	10,000	7.199	–	22,950	91,572	–
	100,000	9.938	–	68,967	585,654	–
	200,000	10.273	–	91,938	684,090	–
	300,000	10.563	–	114,226	771,598	–
	400,000	10.929	–	135,874	848,866	–
CEP with Uncertainty	10,000	7.803	1.08	22,950	300,931	3.00
	100,000	10.842	1.09	68,967	1,754,746	3.07
	200,000	11.267	1.10	91,938	2,102,941	3.14
	300,000	11.736	1.11	114,226	2,420,401	3.19
	400,000	12.619	1.15	135,874	2,709,061	3.29

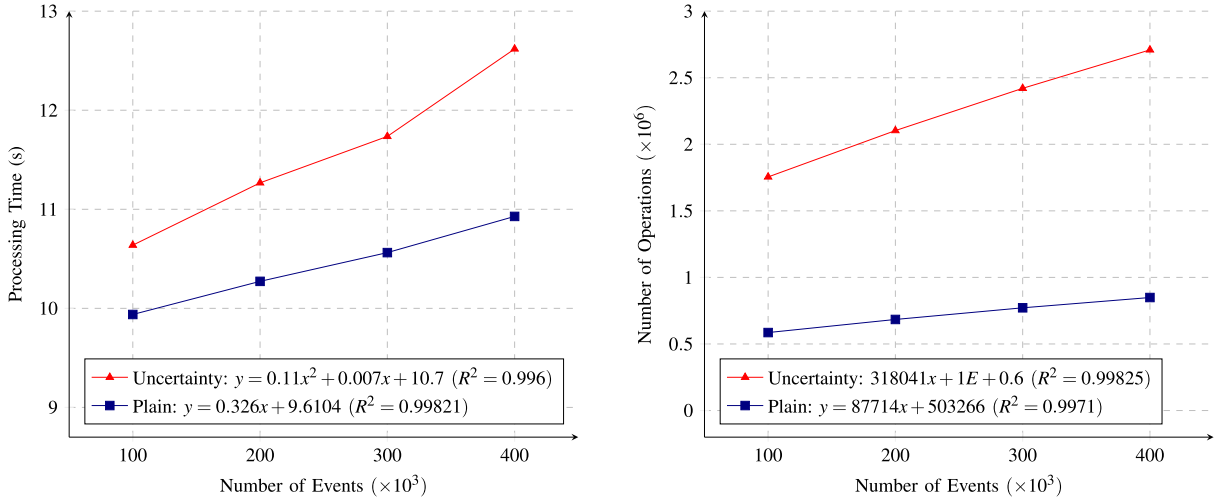

FIGURE 2. Smart House Performance, using Esper: Execution times (left) and Number of operations (right).

complex events), the total number of mathematical and Boolean operations involved, and the overhead caused by the introduction of uncertainty. The corresponding figures accompanying the tables show graphically the differences in execution time and the number of operations between the 'plain' application (i.e, without uncertainty) and the extended one (with uncertainty).

All tests were performed on a MacBook Pro computer, with an Intel Core i7 processor with four 2.8GHz cores, 6 MB shared level 3 cache, 16 GB of onboard 2133 MHz LPDDR3 SDRAM, and 256 GB of onboard flash storage. Execution times were obtained by averaging 5 consecutive runs of each program, after removing the first two executions to avoid the warm-up periods. Given the sample sizes, with five consecutive runs we could ensure a 95% confidence level and an accuracy of 5% of the resulting Fig. [50].

a: THE SMART HOUSE CASE STUDY

Fig. 2 shows a table with the performance figures obtained for the smart house application, depending on the number of processed events, together with the graphical representation of the results. Note that the performance of this application is linear with respect to the number of input events (i.e., simple events).

The overhead in this case is between 1.08 and 1.15. We can see how the number of rules is the same in both implementations because the measurement uncertainty of the input events does not have any effect on the rules' selection patterns (the same applies to every application). The overhead in the operations is on average 3.14, since they need to be performed not only during the selection phase, but also during the calculation of the probabilities of the complex events generated by the rules.

b: THE TUNNEL VENTILATION SYSTEM

Fig. 3 (upper part) shows a table with the performance figures for this application, depending on the number of processed events. These results are shown graphically in the lower part of Fig. 3. One of the main characteristics of this CEP application is that it uses a `not` operator in one of the patterns (the second one). This has an effect on its performance, which grows polynomially (x^2) with the number of events. Accordingly, the performance (and hence the overhead) of the CEP application with uncertainty is worse than in the previous example. Note that in this case, and for the number of events considered, the growth of the execution time of the extended application can be approximated by a linear function, which is not the case for the plain CEP application.

The overhead in this case is between 1.01 (for a small number of input events) and 3.48 (for 200,000 events).

c: AIR QUALITY MONITORING SYSTEM

Fig. 4 shows, both numerically and graphically, the performance results for the Air Quality Monitoring application, depending on the number of incoming events. Although the

		Performance				
	#Events	Time (s)	Ovhd.	#T.Rules	#Opers.	Ovhd.
Plain CEP	10,000	4.031	–	3,130	19,060	–
	100,000	15.479	–	31,300	190,600	–
	200,000	55.725	–	62,600	381,200	–
	300,000	145.441	–	93,900	571,800	–
	400,000	288.982	–	125,200	762,400	–
CEP with Uncertainty	10,000	4.091	1.01	3,130	36,290	1.90
	100,000	46.402	2.99	31,300	362,900	1.90
	200,000	196.230	3.52	62,600	725,800	1.90
	300,000	375.349	2.58	93,900	1,088,700	1.90
	400,000	548.893	1.90	125,200	1,451,600	1.90

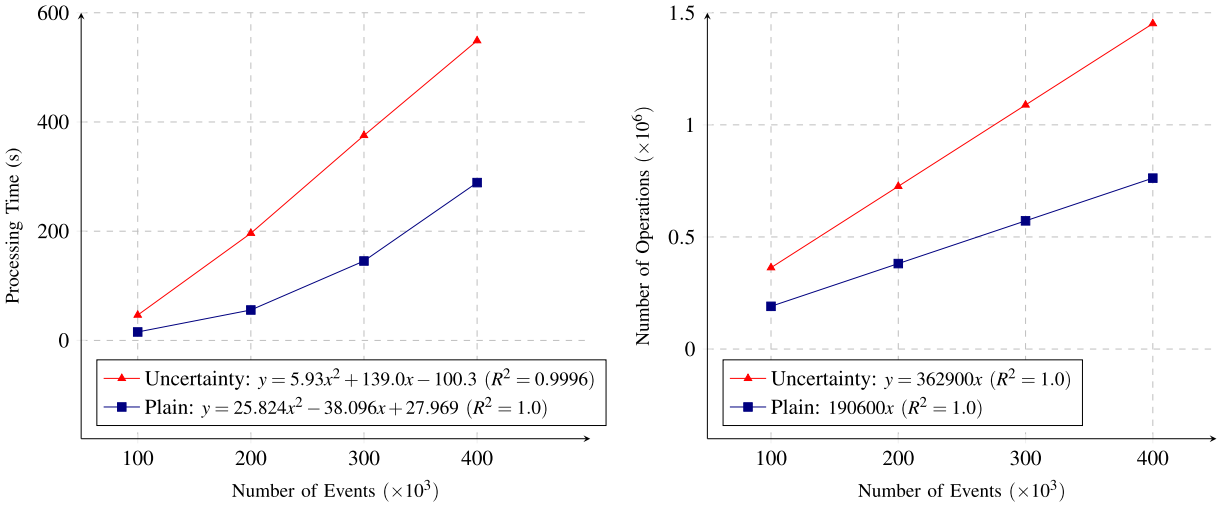

FIGURE 3. Tunnel Ventilation System Performance, using Esper: Execution times (left) and number of operations (right).

operations overhead is constant (2.60, because operations are calculated in two occasions: one for obtaining the values of the complex event and another for obtaining its probability), the overhead of the time performance is higher than that of the Tunnel Ventilation System application. Note that the number of rules that are triggered for every simple event (and hence the number of generated complex events) is much higher in this case (2 million vs. 150K), as well as the average number of operations performed per event (6.99 vs 1.59). The overhead is much higher in this application than in the previous one, although the overhead of uncertainty still grows linearly.

d: A FLEET OF MOTORBIKES

Fig. 5 shows the performance figures for this application, both numerically and graphically. The overhead is between 1.8 and 2.7.

In comparison with the Air Quality system, this application has less rules (only 6) and generates less events (basically, a half), but the number of operations is much higher: roughly, 8.5 more operations per event in the plain CEP application, and 34 in the extended application. In any case, the overhead of the application processing time is maintained below 2.72, and the growth with the number of events is linear (Fig. 5). Another particular aspect of this application is that it does not use uncertainty in the timestamps of the events. Therefore, we decided to use the timestamps provided by the CEP engine (`current_timestamp`) instead of those in the events.

e: THE MADRID 2018 MARATHON

Fig. 6 shows the performance figures obtained for this application, depending on the number of incoming events. For this case study, we used the data provided by the organization of the Madrid Marathon.[6] A total of 9,085 runners finished the marathon, producing a total of 90,850 simple events. To analyze the performance of the application with more events (we wanted to go up to 400,000), we duplicated the source data up to four times. Uncertainty was artificially added to all net times, assuming a clock accuracy of ± 1 second.

This application is the most demanding from all the case studies presented in this paper. It contains two patterns with a negative condition (a `not` operation), and the numbers of triggered rules and operations are much larger than those in the previous examples. As shown in Fig. 6, for 400,000 incoming events the application generates almost 16 million complex events and performs more than 188 million (w/o uncertainty), or 330 million (with uncertainty), operations. The performance is reasonable since the plain CEP application takes 8.2 seconds to process all the events produced by the

[6] www.runrocknroll.com/madrid/en/

		Performance				
	#Events	Time (s)	Ovhd.	#T.Rules	#Opers.	Ovhd.
Plain CEP	10,000	6.563	–	52,487	87,409	–
	100,000	17.394	–	524,870	874,090	–
	200,000	28.723	–	1,049,740	1,748,180	–
	300,000	40.613	–	1,574,610	2,622,270	–
	400,000	51.425	–	2,099,320	3,495,880	–
CEP with Uncertainty	10,000	12.246	1.86	52,487	227,305	2.60
	100,000	57.353	3.30	524,870	2,273,050	2.60
	200,000	107.437	3.74	1,049,740	4,546,100	2.60
	300,000	157.349	3.87	1,574,610	6,819,150	2.60
	400,000	209.852	4.08	2,099,320	9,119,240	2.60

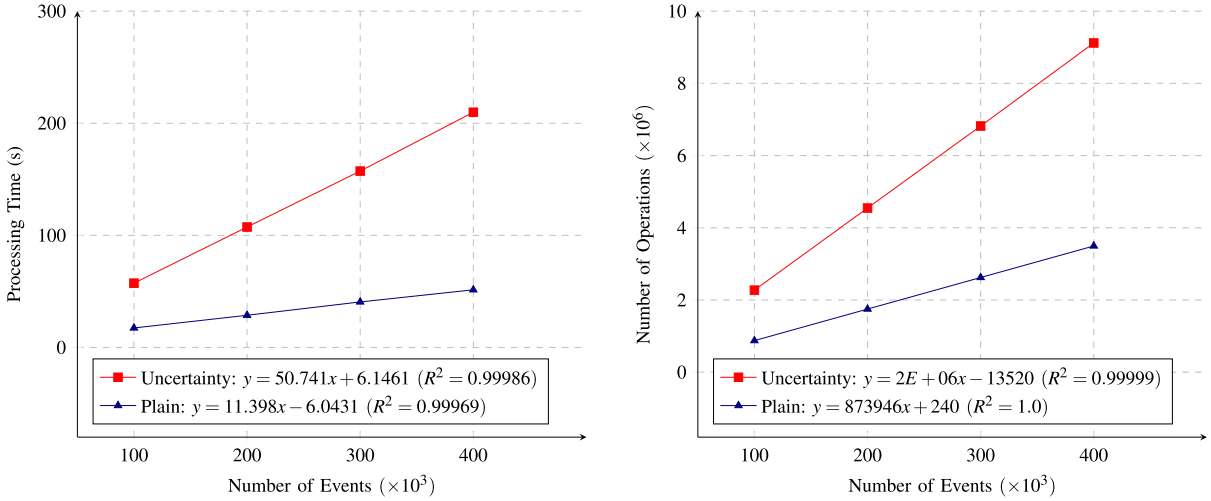

FIGURE 4. Air Quality Performance, using Esper: Execution times (left) and number of operations (right).

Marathon (close to 100,000) and only 13.8 seconds if uncertainty is added to the events. Besides, Fig. 6 also shows that in this case the execution time and the number of performed operations by both the plain and the extended CEP applications do not follow a linear function, but a polynomial one.

2) PERFORMANCE OVERHEAD WHEN INTRODUCING SYNTHETIC RULES

The addition of synthetic rules to capture false negatives in CEP applications has an impact in the performance too. We evaluate here the overhead that these rules introduce.

We have implemented three synthetic rules in the Motorbike CEP application, where it makes more sense to consider false negatives due to the loss of events, as discussed in Sect. III-G. These three new synthetic rules are in charge of dealing with the loss of critical events in the rules that detect that the driver left the seat (`DriverLeftSeat`), the speed suddenly decreased (`Crash`), or that the tires lost pressure (`BlowOutTire`). This produces a CEP system with 10 rules, instead of the seven rules that it originally had. An example of one of these rules was shown in Sect. III-G, when introducing the concept of synthetic rules.

Table 1 shows the performance overhead caused by the introduction of the new three rules, both with and without considering uncertainty. The overhead is in average 1.08, and the median is 1.06, which, in our opinion, is acceptable. It is important to highlight that the rules we have defined use a small temporal window in order to minimize the impact of the `not exists` operator on the stream. This operator can have a significant impact on the performance of the rules when carelessly applied [21].

D. ACCURACY

In this section we are interested in evaluating whether the use of uncertainty in CEP events and rules provides more accurate results to the CEP system users, i.e., whether it pays off regarding the accuracy of the results of the system. We will follow an evaluation schema similar to that of Cugola [21].

We start with a *perfect* system which we assume correct, i.e., the set of input events (S) represent true (theoretical) values, without measurement error or data uncertainty, and the set of complex events that the CEP system generates (O) represent the correct results. These will serve as our oracle.

We also define the precision of the attributes of the simple events. For this, we use the information available about the sensors and clocks used in the applications, and we enrich the values of the set S with their corresponding uncertainty, as we did in Sect. IV-B when checking the correctness of our extension. Let us call $\tilde{S}_1$ to the set whose events are endowed with uncertainty, and $\tilde{O}_1$ to the corresponding set of output events. We already know (cf. Sect. IV-B) that if we process this enriched set $\tilde{S}_1$, the resulting events will be the same

		Performance				
	#Events	Time (s)	Ovhd.	#T.Rules	#Operations	Ovhd.
Plain CEP	10,000	4.134	–	15,872	45,637	–
	100,000	10.058	–	256,064	827,027	–
	200,000	19.122	–	512,291	1,668,555	–
	300,000	25.145	–	778,777	2,514,221	–
	400,000	30.979	–	1,032,622	3,361,231	–
CEP with Uncertainty	10,000	7.549	1.83	15,872	189,366	4.15
	100,000	25.192	2.50	256,064	3,355,002	4.06
	200,000	44.516	2.33	512,291	6,774,470	4.06
	300,000	64.573	2.57	778,777	10,194,783	4.05
	400,000	84.243	2.72	1,032,622	13,641,946	4.06

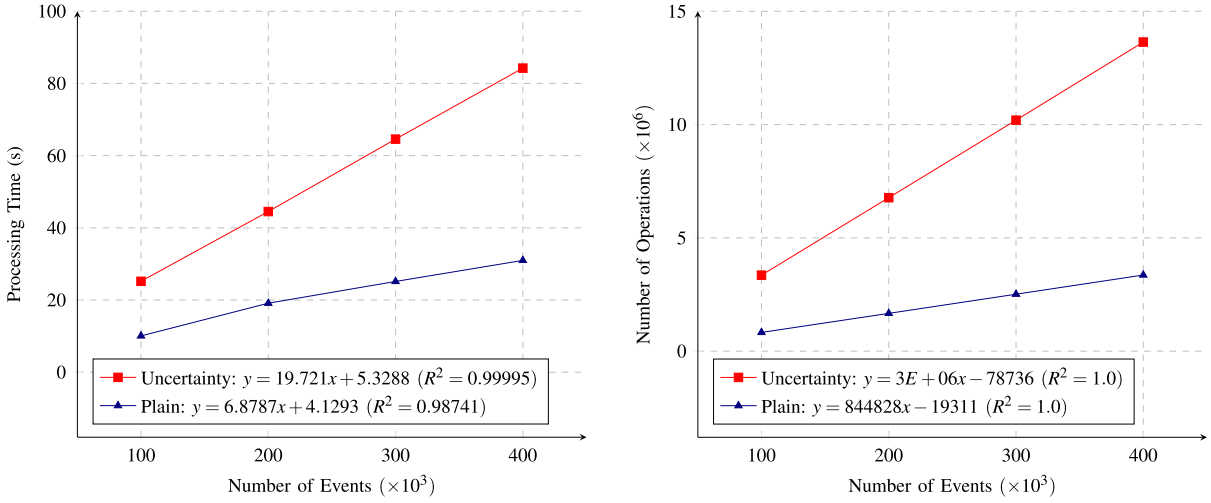

FIGURE 5. Motorbike System Performance, using Esper: Execution times (left) and number of operations (right).

TABLE 1. Motorbike System Performance with Synthetic rules.

#Events	Performance (seconds) and Overheads					
	Plain CEP	Plain CEP +Synth.R.	Ovhd.	CEP +Unc.	CEP+Unc. +Synth.R.	Ovhd.
10,000	4.134	5.302	1.28	7.549	8.269	1.09
100,000	10.058	11.466	1.14	25.192	26.673	1.06
200,000	19.122	19.882	1.04	44.516	45.034	1.01
300,000	25.145	26.076	1.04	64.573	65896	1.02
400,000	30.979	33.900	1.09	84.243	89.758	1.06

as of O, but now with extra information about the probability of their occurrence and the aggregated measurement uncertainty in the values of their attributes. Obtaining this new set ($\tilde{O}_1$), instead of with O, already provides interesting benefits, because it enables CEP users to identify the degree of confidence they can have on the resulting events, and make more informed decisions.

To analyze the accuracy of our proposal, based on the events of the set S and the nominal values of the measurement uncertainty, we generate a new set S_2 of input events with imprecise values using sample generators, assuming a Normal distribution for the uncertainty values [36]. They try to represent the actual events that we would obtain when operating in an environment with uncertainty and imprecise measuring tools. Then, we follow the same process as above: the set S_2 is processed by the CEP engine, producing the corresponding set of complex events O_2 (without uncertainty) and, in parallel, a set $\tilde{S}_2$ is generated by enriching S_2 with uncertainty (using the same precision we have used to generate the values of $\tilde{S}_1$). The processing of $\tilde{S}_2$ produces a new set of complex events $\tilde{O}_2$. Notice that $\tilde{O}_1$ contains the same events as the oracle O, and that $\tilde{O}_2$ contains the same events as O_2 (because our solution respects the set of resulting events, only adding confidence to the events and measurement uncertainty to the values of their attributes), but O and O_2 may be different, and hence $\tilde{O}_1$ and $\tilde{O}_2$ may differ, too. Then, $\tilde{O}_1$ and $\tilde{O}_2$ can be compared, looking for: *true positives* (*TP*), which are composite events appearing in both sets; *false positives* (*FP*), events in $\tilde{O}_2$ but not in $\tilde{O}_1$; and *false negatives* (*FN*), i.e., those events in $\tilde{O}_1$ but not in $\tilde{O}_2$.

		Performance				
	#Events	Time (s)	Ovhd.	#T.Rules	#Operations	Ovhd.
Plain CEP	10,000	3.500	–	2,491	18,004	–
	100,000	8.266	–	315,091	3,648,433	–
	200,000	15.880	–	2,142,281	25,409,263	–
	300,000	31.704	–	6,850,571	817,10,493	–
	400,000	57.467	–	15,808,961	188,980,123	–
CEP with Uncertainty	10,000	3.557	1.02	2,491	32,105	1.78
	100,000	13.261	1.60	315,091	6,397,877	1.75
	200,000	50.791	3.19	2,142,281	44,509,047	1.75
	300,000	137.095	4.32	6,850,571	143,082,517	1.75
	400,000	293.042	5.10	15,808,961	330,867,287	1.75

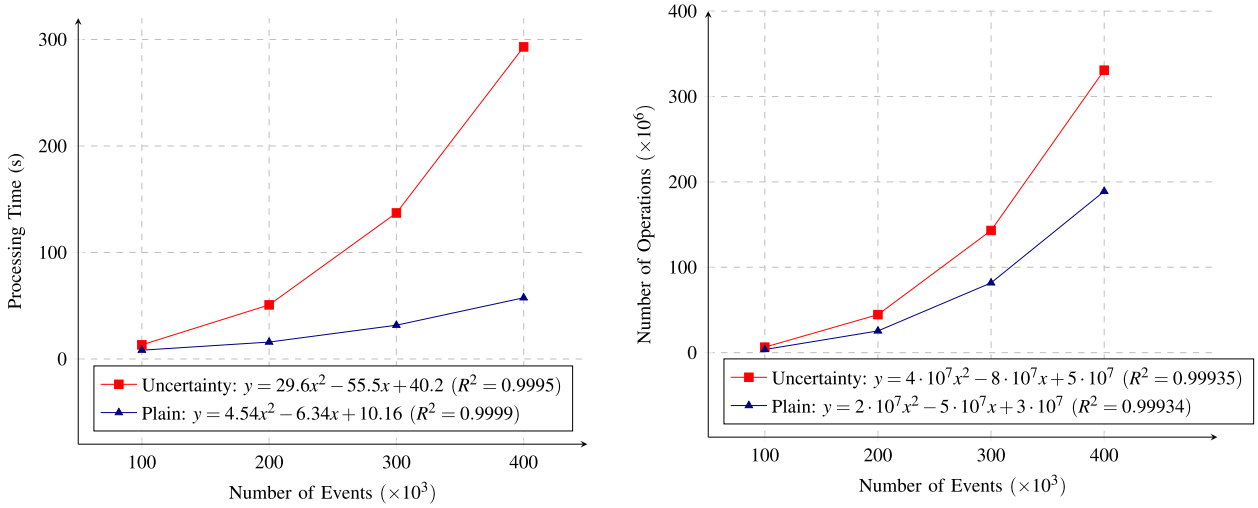

FIGURE 6. Marathon Application Performance, using Esper: Execution times (left) and number of operations (right).

Given that events are periodically generated, *true negatives* (*TN*) are those events that do not appear in any of the two sets.

With this, we can define the Precision $P = TP/(TP + FP)$ and Recall $R = TP/(TP+FN)$ of the execution, as well as its Accuracy $A = (TP + TN)/(TP + TN + FP + FN)$. We study these metrics for different kinds of rules: selection, combination and hierarchy, taken from the smart house example.

a: SELECTION

The CO High Level is an example of a selection rule. It creates a `COHigh` event every time a simple `HomeEvent` is detected whose CO level exceeds 5,000 units. We created 54 different sets of input events, whose results are shown in Figs. 7 and 8. For each test, we generated 20,000 simple events, half of them with a CO level below the threshold of 5,000 CO units (the value that triggers the rule), and the other half with a value above that threshold. The ranges of the CO levels of the input events varied, depending on how much dispersed, or concentrated, they were. For example, for the set of 20,000 input events whose CO level ranged between 2,000 and 8,000 units (indicated as [2-8] on the x-axis of Figs. 7 and 8), we generated 10,000 events with a CO value between 2,000 and 5,000, and the other 10,000 events with values between 5,000 and 8,000. Thus, we always exceed the threshold level in half of the generated events. Every one of the 18 sets of varying ranges (shown in the x-axis) was produced using a different measurement uncertainty for the CO level attribute: 0.1, 0.25 and 0.5. The degree of uncertainty has influence on the accuracy and precision of the results. When values are spread over a wider interval, the accuracy of the CEP system is high because the values are easy to tell apart. The accuracy starts decreasing as values concentrate around the threshold, because numbers with uncertainty are difficult to differentiate when they become sufficiently close to each other—until they become so close to the threshold that

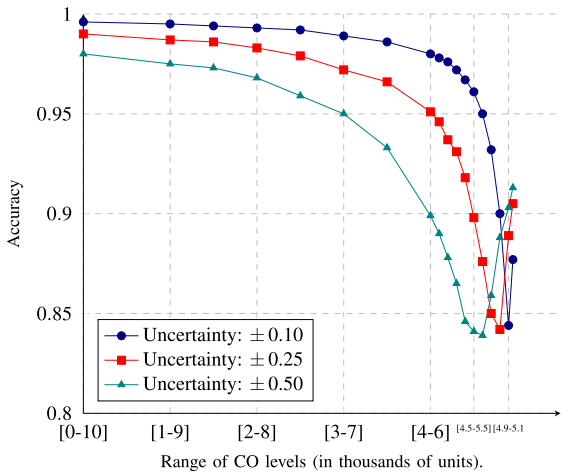

FIGURE 7. Accuracy of rule `COHigh` for different uncertainties.

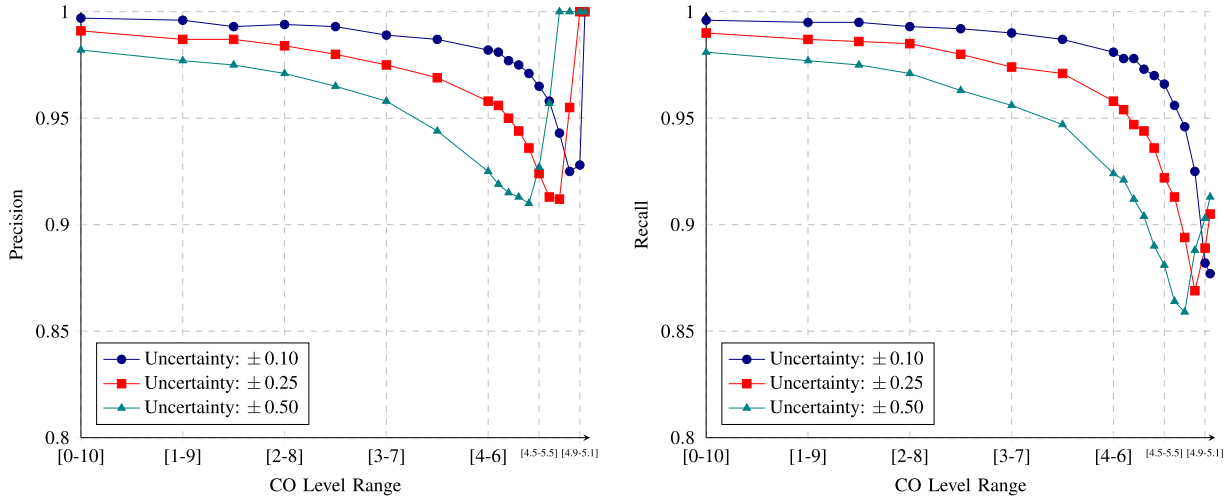

FIGURE 8. Precision and Recall of rule `COHigh` for different uncertainties.

they become almost indistinguishable, and hence the accuracy grows again. In any case, the accuracy is always above 0.8, and in most cases it remains above 0.9. These results are completely on par with the ones obtained by Cugola [21] for the similar experiment, hence replicating his conclusions.

Fig. 8 shows the results for the precision and recall of the CO High level *selection* rule. They are similar to those obtained for the accuracy: values far from the trigger threshold of the rule obtain close to 100% precision and recall; however, both measures decrease when events are grouped with values close to the threshold. In any case, precision always remains above 90%, which indicates a small percentage of false positives, while recall is always above 85%.

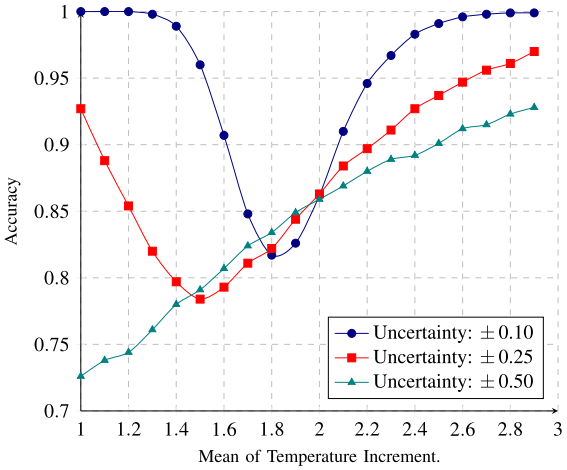

FIGURE 9. Accuracy of rule `TempIncrease`.

b: COMBINATION

Fig. 9 shows the results for an event combination rule, namely the Temperature Increase rule, which is triggered when a 2-degree increase is detected between two consecutive events. The temperature values of the 20,000 events generated in each test ranged between 28 and 42 degrees with temperature increments from 1.0 to 2.9. The closer the increment is to the threshold value of 2, the higher the imprecision is, specially when the uncertainty grows too. In this case, the accuracy of the rules with uncertainty is similar to the previous one, although with slightly smaller values when the uncertainty is large (0.5).

Fig. 10 shows the precision and recall values for a combination rule. When temperature increases are very low, i.e., relatively far from the threshold of 2, almost all events (19.995) are TN, with only a few FP events. Therefore, the precision is almost nil. Conversely, when the values exceed the threshold of 2.0, most events are TP and the precision becomes close to 1.0. The Recall behavior is similar, although less pronounced due to a lower occurrence of FN events.

c: HIERARCHY

Finally, we have analyzed a hierarchy rule that depends on other complex events (Figs. 11 and 12). In this case, we use the Temperature Warning rule, which is triggered when 4 complex Temperature Increase events arrive whose temperature values are above a threshold of 35 degrees. As it is dependent on the previous rule, the resulting curves are similar, although in this case with better accuracy values (Fig. 11). Somehow, the first rule in the hierarchy filters the wrong values. This is also the reason why the precision and recall results are similar (Fig. 12).

E. APPLICABILITY

To assess the applicability of our proposal, we tried to evaluate how much effort is needed to apply our approach to existing CEP languages and/or engines.

Apache Flink is an open source stream processing framework, similar to Spark, that provides a distributed streaming dataflow engine written in Java and Scala. It works in both batch and pipelined modes, using a directed graph approach that leverages in-memory storage for significant performance gains. We have used Flink 1.7 and its CEP library to implement some of the case studies presented in this paper. Due to the open source nature of Flink, and the way CEP rules are

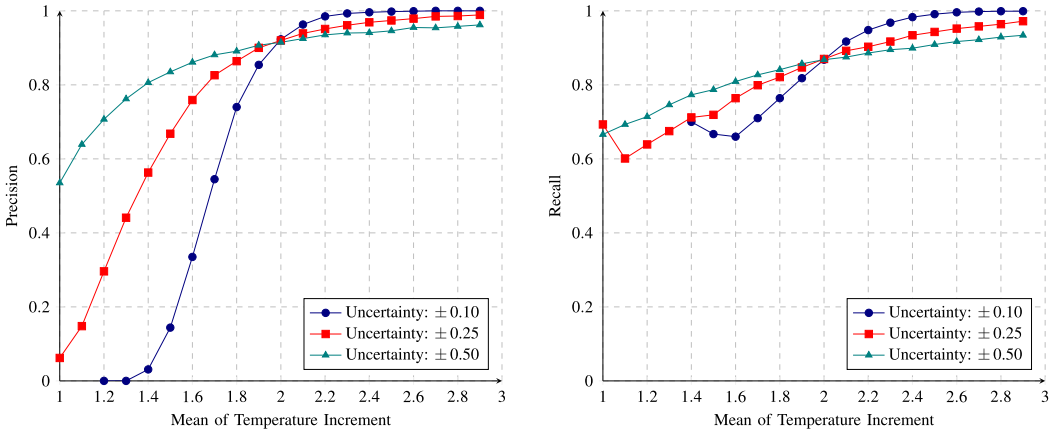

FIGURE 10. Precision and Recall of rule `TempIncrease` for different uncertainties.

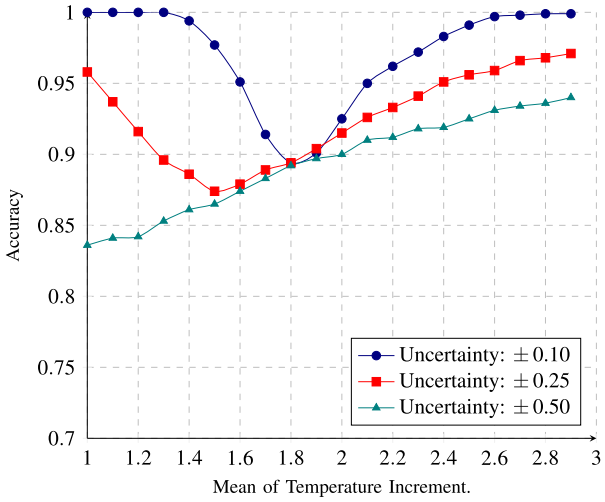

FIGURE 11. Accuracy of rule `TempWarning`.

written in Flink, adding uncertainty to the CEP events and rules using our library of uncertain types is straightforward.

For example, in the Smart House case study, we have defined the simple event `UHomeEvent`, where several fields are declared with uncertainty using the extended types `UReal`, `UBoolean` and `UInteger`:

```
public class UHomeEvent extends SimpleEvent{
    private final UReal temp;
    private final UReal coLevel;
    private final UBoolean doorOpen;
    private final UReal coorX;
    private final UReal coorY;
    private final double sqre;
    private final UInteger ts;
    . . . . .
}
```

These events are used normally in the rules, processing them as other events without uncertainty. Uncertain values are operated using the methods defined for the corresponding extended types. For example, the following listing defines the rule `TemperatureIncrease` in Apache Flink, which generates a complex event `UTempIncreaseEvent` whose attributes are also endowed with uncertainty. In this way, uncertainty spreads through the hierarchy of rules in a natural way. Notice that CEP rules in Flink are defined in three steps: first we define the pattern, then the stream to deal with the input events, and finally we define how and when the complex events produced by the rule are generated:

```
// Create a Pattern
Pattern<SimpleEvent,?> tempIncreasePattern =
  Pattern.<SimpleEvent>begin("first")
  .subtype(UHomeEvent.class)
  .followedBy("second")
  .subtype(UHomeEvent.class)
  .within(Time.seconds(5L));
// Create pattern stream for warning pattern
PatternStream<SimpleEvent>tempPatternStream=
  CEP.pattern(partitionedEventStream,
              tempIncreasePattern);
// Generate temp. warning complex events
DataStream<UTempIncreaseEvent>tempIncreases=
  tempPatternStream.flatSelect(
  new PatternFlatSelectFunction<SimpleEvent,
      UTempIncreaseEvent>() {
@Override
public void flatSelect(Map<String,
  List<SimpleEvent>> pattern,
  Collector<UTempIncreaseEvent> collector)
    throws Exception {
  UHomeEvent first  =
   (UHomeEvent)pattern.get("first").get(0);
  UHomeEvent second =
   (UHomeEvent)pattern.get("second").get(0);
  UReal increment =
   second.getTemp().minus(first.getTemp());
   if(increment.ge(new UReal(2.0, DELTA)).
      toBoolean()){
   collector.collect(new UTempIncreaseEvent(
        first.getHomeID(), second.getTemp(),
        increment, second.getTs()));
   }
 }
});
```

Table 2 shows the performance figures we obtained for the Smart House case study using Apache Flink. Although slightly slower than Esper (cf. Fig. 2), the overhead introduced by the use of uncertainty in the events and in the

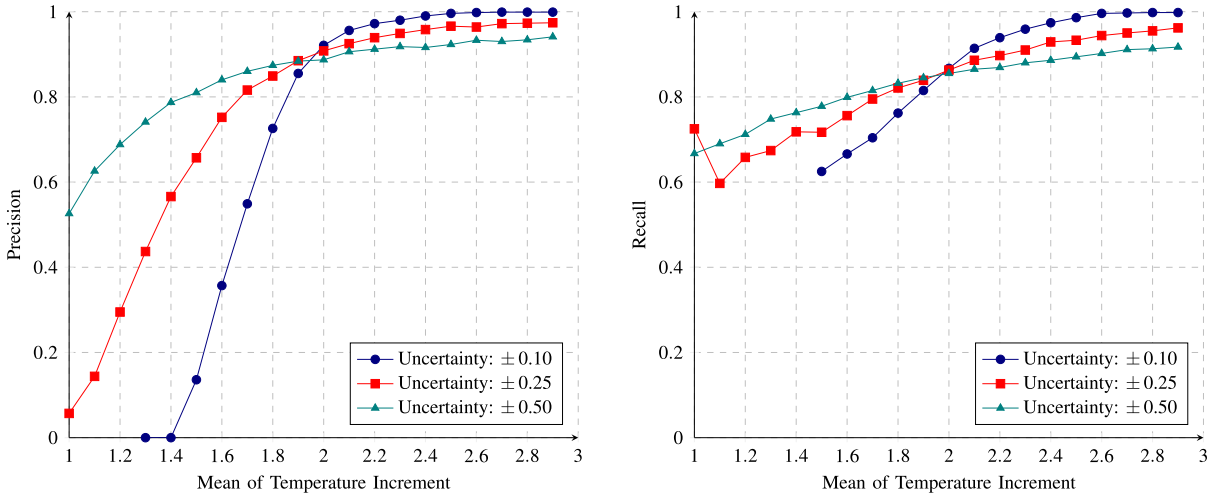

FIGURE 12. Precision and Recall of rule `TempWarning` for different uncertainties.

TABLE 2. Smart house performance using Apache Flink.

#Events	Performance (seconds) and Overheads		
	Plain CEP	CEP w/Uncert.	Overhead
10,000	4.3	4.4	1.024
100,000	28.7	29.0	1.011
200,000	90.5	91.9	1.016
300,000	144.7	147.6	1.020
400,000	199.8	202.4	1.013

rules has improved: between 1.011 and 1.024 in this case, with an average of 1.7% with Apache Flink, and of around 10% with Esper.

F. LIMITATIONS AND POTENTIAL LINES FOR IMPROVEMENT

This final subsection describes some of the limitations that we have found during the evaluation process of our proposal, as well as some potential lines of work to improve them.

- In principle, Esper does not seem to allow for a library of types to be used in its aggregation functions (`count`, `max`, `sum`, etc.). This means that the Esper source code needs to be directly changed if these functions have to operate with uncertain values—e.g., by using our library. Although there is no particular problem with this, since Esper is an open source software, it would be better if this feature could be implemented in the open source distribution version of Esper. This limitation does not apply to Apache Flink, because its language is completely flexible.
- Under the presence of uncertainty, in order to calculate the values of the complex events produced by the CEP rules, and of their probability, many operations need to be performed twice: one to compute the values and one to compute the confidence. We are working for optimization of our implementation in Esper that permits calculating both results at the same time, whenever this is possible.
- We have realized that adding uncertainty to the rules of an existing CEP application is in general conceptually simple, but rather tedious and error-prone. Therefore, it should be better automated, so human intervention is not required (apart from defining the probabilities of the CEP rules). In this respect, we would like to develop a tool that is able to automatically generate the code of the CEP patterns and rules extended with uncertainty, using Model-Driven Engineering techniques, in a similar way to the tool we developed for the static analysis of CEP applications [51].
- The SQL-style of Esper makes easier and more natural the specification of the CEP rules, in contrast with the open API style of Flink. Nevertheless, we would need to evaluate such a claim and support it with empirical evidences
- While measurement and occurrence uncertainty are static during the early lifetime of sensors and other physical devices, uncertainties can rapidly deviate from those stated in the data sheets when in the field, especially with low-quality sensors, enclosures, and wireless networks. This requires complex models of uncertainty to represent the dynamic and transient nature of uncertainty in these situations. Although these models can be a priori implemented with our library, their detailed analysis requires its own dedicated line of research.
- In this paper, we have used event time semantics to reason about time. However, other time semantics could be able to express different uncertainties, such as event reordering or imprecise clocks [32]. Dealing with different kinds of time semantics [33] would be another interesting line of future work.

V. RELATED WORK

Several authors have acknowledged the need to handle uncertainty in CEP systems, and this has been an active line of research since 2008. The work by Alevizos *et al.* [22] provides a very complete survey on the proposals that try to address uncertainty in CEP systems. In general, each proposal has focused on a particular kind of uncertainty, from the phases identified above in Sect. II-C.

A first set of works focuses on the uncertainty in the selection of the antecedents, assigning probabilities to events. These probabilities represent the confidence we have on their occurrence. For example, Ré *et al.* [52] extend the Cayuga engine [29], [53] with an efficient inference mechanism for answering queries over probabilistic data streams of events tagged with probabilities, assuming the events follow a first-order Markov process. Kawashima *et al.* [54] extend the SASE+ engine [55] by building a deterministic automaton for every CEP pattern, being able to detect patterns above a certain confidence threshold by developing a matching tree as new events arrive, and until the time window of the query expires. Xu *et al.* [56] propose an Instance Pruning and Filter-Detection Algorithm (IPF-DA) over probabilistic data streams, able to detect complex events satisfying the patterns with a single scanning of the event stream. In addition, they build a Bayesian network to express and infer the probability distribution of uncertain events. A later work by Wang *et al.* [20] extends this proposal with the ability to express event hierarchies, i.e., higher-order events.

In all these works, uncertainty is restricted to the confidence on the events, while the values of the attributes are all certain. This includes events timestamps. Up to our knowledge, only one work [43] deals with uncertain timestamps, although all the other attributes have crisp values. In that work, timestamps follow a known probability distribution (usually Uniform), and the authors propose an efficient algorithm for constructing event matching and their intervals without the need to enumerate all possible worlds [22]. This method only supports sequence patterns with simple equality/inequality predicates, but it was extended in [57] to deal with negation and user-defined predicates.

Cugola *et al.* [21] extend TESLA [27], [58] with probabilistic modeling, in order to handle the uncertainty both in events and in CEP rules. In their proposal, called CEP2U and implemented over T-REX [16], events are endowed with probabilities that indicate the degree of confidence in their occurrence. This approach also permits associating uncertainty to event attributes, which are modeled as random variables with some measurement error. The error follows a known probability distribution function, e.g., Gaussian. Furthermore, using an open-world approach, CEP patterns are associated a degree of uncertainty because they can be based on incomplete or erroneous assumptions about the environment of the system. Therefore, the method automatically builds a Bayesian Network for each pattern. The probabilistic parameters of the network are manually estimated by domain experts. As known limitations, CEP2U does not deal with imprecise timestamps, and the propagation of uncertainty in the calculation of the attributes' values needs to be computed by the user.

Another approach that permits dealing with uncertainty in the occurrence of events and in the CEP patterns is called KBMC, and is due to Wasserkrug *et al.* [19]. They use Bayesian Networks for addressing both kinds of uncertainties, but rather differently from CEP2U. In particular, KBMC defines only one Bayesian Network for the complete system (including its patterns and events), which is built as simple events arrive. Each event is assigned a probability, denoting how probable it is that the event occurred with specific values for its attributes. The nodes of the network correspond to events, both simple and complex. This method is far more complex and computationally expensive than CEP2U, although in order to increase performance, the latter makes some significant assumptions. For example, CEP2U assumes that a complex event cannot be defined by multiple rules, but a primitive event can participate in multiple rules. In addition, occurrence probabilities of the intermediate events (i.e., the antecedents) are propagated to their corresponding complex event with a value of 1, which means that the Bayesian Networks function more like look-up tables, hence the much lower cost of inference [22].

In our approach, we depart from these approaches in several ways. First, we use the extended type system for Real numbers and Boolean values defined in [23]. This greatly simplifies the representation of the uncertainty and its propagation through the operations. Furthermore, we separate the process of calculating the probability of every CEP rule from its application. In this way, we use a variable (that can be a constant or just a function) that determines the probability of a rule, independently from how such variable is calculated: either using Bayesian Networks, such as in [19] or in [21], or by any other means. Finally, we deal with uncertain timestamps in a natural way by representing them using uncertain integers (type `UInteger`). This permits associating a confidence level to all comparisons and operations on timestamps in a transparent manner, and without requiring any user computation.

VI. CONCLUSIONS AND FUTURE WORK

In this paper, we have presented a proposal to deal with measurement and occurrence uncertainty in CEP systems, particularly those that analyze streams of real-time events coming from physical sources. Different kinds of uncertainties have been identified and incorporated into the events and CEP rules, allowing modelers to represent and manage this kind of information. We do not propose yet another CEP engine, but a library that can be added to existing ones, together with a method to use it.

The proposal has been validated with several applications that have served to assess the performance, expressiveness and accuracy of our approach, and to identify both strong

points and issues that would require future improvements. These applications have been developed on top of the Esper and Apache Flink CEP engines, and showed good results.

The examples have also served to illustrate the need to consider uncertainty and probability in CEP systems. For example, instead of all events being equally probable, our proposal permits assigning confidence to events. This introduces an implicit prioritization mechanism, very useful for instance to discard events which are very unlikely to happen, or to determine the order in which we should be dealing with two or more critical events according to the relative confidence we have on their occurrence. The costs in terms of the addition of uncertainty to the rules, as well as the performance overheads, do not seem to significantly hinder our proposal.

As part of our future work, we plan to address the issues identified in Section IV-F. We would also like to widen the number of case studies and apply our proposal to more CEP applications from different domains, hence gaining more experience on its applicability and use.

ACKNOWLEDGMENTS

The authors would like to thank Prof. Gianpaolo Cugola for his help setting up and running our TESLA environment, and for his detailed responses to all our questions. They would also like to thank the reviewers for their insightful and helpfulp comments and suggestions on previous versions of this manuscript. N. Moreno, M. F. Bertoa, and A. Vallecillo were with the Departamento de Lenguajes y Ciencias de la Computación, Universidad de Málaga, Málaga, Spain. L. Burgueño was with IN3, Open University of Catalonia, Barcelona, Spain, and also with the Institut LIST, CEA, Université Paris-Saclay, Paris, France.

REFERENCES

[1] C. Zang and Y. Fan, "Complex event processing in enterprise information systems based on RFID," *J. Enterprise Inf. Syst.*, vol. 1, no. 1, pp. 3–23, 2007.

[2] K. Broda, K. Clark, R. Miller, and A. Russo, "SAGE: A logical agent-based environment monitoring and control system," in *Proc. AmI*, in Lecture Notes in Computer Science, vol. 5859. Berlin, Germany: Springer, 2009, pp. 112–117.

[3] H. Macià, V. Valero, G. Díaz, J. Boubeta-Puig, and G. Ortiz, "Complex event processing modeling by prioritized colored Petri nets," *IEEE Access*, vol. 4, pp. 7425–7439, 2016.

[4] A. Y. Sun, Z. Zhong, H. Jeong, and Q. Yang, "Building complex event processing capability for intelligent environmental monitoring," *Environ. Model. Softw.*, vol. 116, pp. 1–6, Jun. 2019.

[5] A. Demers, J. Gehrke, M. Hong, M. Riedewald, and W. White, "Towards expressive publish/subscribe systems," in *Proc. EDBT*, in Lecture Notes in Computer Science, vol. 3896. Berlin, Germany: Springer, 2006, pp. 627–644.

[6] J. Boubeta-Puig, G. Ortiz, and I. Medina-Bulo, "MEdit4CEP: A model-driven solution for real-time decision making in SOA 2.0," *Knowl.-Based Syst.*, vol. 89, pp. 97–112, Nov. 2015.

[7] K. Patroumpas, E. Alevizos, A. Artikis, M. Vodas, N. Pelekis, and Y. Theodoridis, "Online event recognition from moving vessel trajectories," *GeoInformatica*, vol. 21, no. 2, pp. 389–427, 2017.

[8] N. Marz and J. Warren, *Big Data: Principles and Best Practices of Scalable Realtime Data Systems*. Shelter Island, NY, USA: Manning Publications, 2015.

[9] S. Shi, D. Jin, and G. Tiong-Thye, "Real-time public mood tracking of chinese microblog streams with complex event processing," *IEEE Access*, vol. 5, pp. 421–431, 2017.

[10] A. García-de-Prado, G. Ortiz, and J. Boubeta-Puig, "COLLECT: COLLaborativE ConText-aware service oriented architecture for intelligent decision-making in the Internet of Things," *Expert Syst. Appl.*, vol. 85, pp. 231–248, Nov. 2017.

[11] S. Greengard, *The Internet of Things*. Cambridge, MA, USA: MIT Press, 2015.

[12] G. Cugola and A. Margara, "Processing flows of information: From data stream to complex event processing," *ACM Comput. Surv.*, vol. 44, no. 3, pp. 15-1–15-62, 2012.

[13] O. Etzion and P. Niblett, *Event Processing in Action*. Shelter Island, NY, USA: Manning Publications, 2010.

[14] D. C. Luckham, *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems*. Reading, MA, USA: Addison-Wesley, 2002.

[15] D. C. Luckham, *Event Processing for Business: Organizing the Real-Time Enterprise*. Hoboken, NJ, USA: Wiley, 2012.

[16] G. Cugola and A. Margara, "Complex event processing with T-REX," *J. Syst. Softw.*, vol. 85, no. 8, pp. 1709–1728, Aug. 2012.

[17] I. Flouris, N. Giatrakos, A. Deligiannakis, M. N. Garofalakis, M. Kamp, and M. Mock, "Issues in complex event processing: Status and prospects in the Big Data era," *J. Syst. Softw.*, vol. 127, pp. 217–236, May 2017.

[18] A. Artikis, O. Etzion, Z. Feldman, and F. Fournier, "Event processing under uncertainty," in *Proc. ACM DEBS*, 2012, pp. 32–43.

[19] S. Wasserkrug, A. Gal, O. Etzion, and Y. Turchin, "Complex event processing over uncertain data," in *Proc. ACM DEBS*, 2008, pp. 253–264.

[20] Y. H. Wang, K. Cao, and X. M. Zhang, "Complex event processing over distributed probabilistic event streams," *Comput. Math. Appl.*, vol. 66, no. 10, pp. 1808–1821, 2013.

[21] G. Cugola, A. Margara, M. Matteucci, and G. Tamburrelli, "Introducing uncertainty in complex event processing: Model, implementation, and validation," *Computing*, vol. 97, no. 2, pp. 103–144, 2015.

[22] E. Alevizos, A. Skarlatidis, A. Artikis, and G. Paliouras, "Probabilistic complex event recognition: A survey," *ACM Comput. Surv.*, vol. 50, no. 5, pp. 71-1–71-31, 2017.

[23] M. F. Bertoa, N. Moreno, G. Barquero, L. Burgueño, J. Troya, and A. Vallecillo, "Expressing measurement uncertainty in OCL/UML datatypes," in *Proc. ECMFA*, in Lecture Notes in Computer Science, vol. 10890. Cham, Switzerland: Springer, 2018, pp. 46–62.

[24] *Object Constraint Language (OCL) Specification. Version 2.4*, Object Manage. Group, Needham, MA, USA, Feb. 2014.

[25] *Unified Modeling Language (UML) Specification. Version 2.5*, Object Manage. Group, Needham, MA, USA, Mar. 2015.

[26] N. Moreno, M. F. Bertoa, G. Barquero, L. Burgueño, J. Troya, A. García-López, and A. Vallecillo, "Managing uncertain complex events in Web of things applications," in *Proc. ICWE*, in Lecture Notes in Computer Science, vol. 10845. Cham, Switzerland: Springer, 2018, pp. 349–357.

[27] G. Cugola, A. Margara, M. Pezzè, and M. Pradella, "Efficient analysis of event processing applications," in *Proc. ACM DEBS*, 2015, pp. 10–21.

[28] Event Processing Technical Society. (2011). *Event Processing Glossary—Version 2.0*. [Online]. Available: http://www.complexevents.com/wp-content/uploads/2011/08/EPTS_Event_Processing_Glossary_v2.pdf

[29] L. Brenna, A. Demers, J. Gehrke, M. Hong, J. Ossher, B. Panda, M. Riedewald, M. Thatte, and W. White, "Cayuga: A high-performance event processing engine," in *Proc. ACM SIGMOD*, 2007, pp. 1100–1102.

[30] D. Anicic, S. Rudolph, P. Fodor, and N. Stojanovic, "Stream reasoning and complex event processing in ETALIS," *Semantic Web*, vol. 3, no. 4, pp. 397–407, 2012.

[31] *The Open Group Base Specifications. Issue 7, Sect. 4.16, Seconds Since the Epoch*, IEEE Standard 1003.1-2008, 2016.

[32] T. Akidau, R. Bradshaw, C. Chambers, S. Chernyak, R. Fernández-Moctezuma, R. Lax, S. McVeety, D. Mills, F. Perry, E. Schmidt, and S. Whittle, "The dataflow model: A practical approach to balancing correctness, latency, and cost in massive-scale, unbounded, out-of-order data processing," *Proc. VLDB Endowment*, vol. 8, no. 12, pp. 1792–1803, 2015.

[33] L. Affetti, R. Tommasini, A. Margara, G. Cugola, and E. D. Valle, "Defining the execution semantics of stream processing engines," *J. Big Data*, vol. 4, p. 12, Dec. 2017.

[34] L. Burgueño, J. Boubeta-Puig, and A. Vallecillo, "Formalizing complex event processing systems in Maude," *IEEE Access*, vol. 6, pp. 23222–23241, 2018.
[35] EsperTech. *Esper—Complex Event Processing*. Accessed: Mar. 5, 2019. [Online]. Available: http://www.espertech.com/esper/
[36] *Evaluation of Measurement Data—Guide to the Expression of Uncertainty in Measurement (GUM)*, document JCGM 100:2008, Joint Committee for Guides in Metrology, 2008. [Online]. Available: http://www.bipm.org/utils/common/documents/jcgm/JCGM_100_2008_E.pdf
[37] *OMG, UML Profile for Modeling and Analysis of Real-Time and Embedded Systems (MARTE)*, Object Manage. Group, Needham, MA, USA, Jun. 2008.
[38] *Evaluation of Measurement Data—Supplement 1 to the 'Guide to the Expression of Uncertainty in Measurement'—Propagation of Distributions Using a Monte Carlo Method*, document JCGM 101:2008, Joint Committee for Guides in Metrology, 2008. [Online]. Available: http://www.bipm.org/utils/common/documents/jcgm/JCGM_101_2008_E.pdf
[39] M. Zhang, B. Selic, S. Ali, T. Yue, O. Okariz, and R. Norgren, "Understanding uncertainty in cyber-physical systems: A conceptual model," in *Proc. ECMFA*, in Lecture Notes in Computer Science, vol. 9764. Cham, Switzerland: Springer, 2016, pp. 247–264.
[40] *International Vocabulary of Metrology—Basic and General Concepts and Associated Terms (VIM)*, document JCGM 200:2012, 3rd ed., Joint Committee for Guides in Metrology, 2012. [Online]. Available: http://www.bipm.org/utils/common/documents/jcgm/JCGM_200_2012.pdf
[41] B. de Finetti, *Theory of Probability: A Critical Introductory Treatment*. Hoboken, NJ, USA: Wiley, 2017.
[42] B. Kosko, "Fuzziness vs. probability," *Int. J. Gen. Syst.*, vol. 17, nos. 2–3, pp. 211–240, 1990.
[43] H. Zhang, Y. Diao, and N. Immerman, "Recognizing patterns in streams with imprecise timestamps," *Inf. Syst.*, vol. 38, no. 8, pp. 1187–1211, 2013.
[44] W. L. Oberkampf, S. M. DeLand, B. Rutherford, K. V. Diegert, and K. F. Alvin, "Error and uncertainty in modeling and simulation," *Rel. Eng. Sys. Saf.*, vol. 75, no. 3, pp. 333–357, 2002.
[45] N. Moreno, M. F. Bertoa, L. Burgueño, and A. Vallecillo. (2018). *Managing Uncertainty in CEP*. [Online]. Available: http://atenea.lcc.uma.es/projects/UCEP.html and https://github.com/atenearesearchgroup/uncertainCEP.git
[46] J. X. Yu, Z. Chong, H. Lu, Z. Zhang, and A. Zhou, "A false negative approach to mining frequent itemsets from high speed transactional data streams," *Inf. Sci.*, vol. 176, no. 14, pp. 1986–2015, 2006.
[47] L. Antova, C. Koch, and D. Olteanu, "$10^{(10^6)}$ worlds and beyond: Efficient representation and processing of incomplete information," *VLDB J.*, vol. 18, no. 5, pp. 1021–1040, 2009.
[48] E. Letier, D. Stefan, and E. T. Barr, "Uncertainty, risk, and information value in software requirements and architecture," in *Proc. ACM ICSE*, 2014, pp. 883–894.
[49] G. Adzic. (2012). *Redefining Software Quality*. Accessed: Oct. 2018. [Online]. Available: https://gojko.net/2012/05/08/redefining-software-quality/
[50] A. Georges, D. Buytaert, and L. Eeckhout, "Statistically rigorous Java performance evaluation," in *Proc. ACM SIGPLAN Notices (OOPSLA)*, 2007, vol. 42, no. 10, pp. 57–76.
[51] A. García-López, L. Burgueño, and A. Vallecillo, "Static analysis of complex event processing programs," in *Proc. MDE4IoT*, 2018, pp. 1–5.
[52] C. Ré, J. Letchner, M. Balazinska, and D. Suciu, "Event queries on correlated probabilistic streams," in *Proc. ACM SIGMOD*, 2008, pp. 715–728.
[53] A. J. Demers, J. Gehrke, B. Panda, M. Riedewald, V. Sharma, and W. White, "Cayuga: A general purpose event monitoring system," in *Proc. CIDR*, 2007, pp. 412–422.
[54] H. Kawashima, H. Kitagawa, and X. Li, "Complex event processing over uncertain data streams," in *Proc. 3PGCIC*, 2010, pp. 521–526.
[55] J. Agrawal, Y. Diao, D. Gyllstrom, and N. Immerman, "Efficient pattern matching over event streams," in *Proc. ACM SIGMOD*, 2008, pp. 147–160.
[56] X. Chuanfei, L. Shukuan, W. Lei, and Q. Jianzhong, "Complex event detection in probabilistic stream," in *Proc. APWEB*, 2010, pp. 361–363.
[57] H. Zhang, Y. Diao, and N. Immerman, "On complexity and optimization of expensive queries in complex event processing," in *Proc. ACM SIGMOD*, 2014, pp. 217–228.
[58] G. Cugola and A. Margara, "TESLA: A formally defined event specification language," in *Proc. ACM DEBS*, 2010, pp. 50–61.

NATHALIE MORENO received the M.Sc. and Ph.D. degrees in computer science from the Department of Computer Science, University of Málaga, Spain, in 2012, where she is currently an Assistant Professor. Her research interest is mainly oriented toward model-driven development. In particular, she focuses on conceptual modeling methodologies, business process modeling, model transformation languages, and uncertainty on complex event processing systems for its application on the Internet of Things.

MANUEL F. BERTOA received the Ph.D. degree in computer science from the University of Málaga, where he is currently an Assistant Professor. He is also a Telecommunications Engineer with the Technical University of Madrid. He has more than 16 years of experience in international IT companies, public administration, and health care systems. His research interests include software quality, software measurement, and software sustainability.

LOLI BURGUEÑO is currently a Postdoctoral Researcher with the Open University of Catalonia (UOC), Barcelona, Spain, and CEA List, Paris, France. Her research interest includes model-driven engineering (MDE). She has worked on the field of testing model transformations, the distribution of very large models and the parallelization of the execution of model transformations, the formalization of complex-event-processing languages, and the modeling of uncertainty in software models for its use in the Industry 4.0. She is also working on the integration of artificial intelligence techniques into modeling tools and processes. Further information can be found at https://som-research.uoc.edu/loli-burgueno.

ANTONIO VALLECILLO is currently a Professor of software engineering with the University of Málaga, Spain, where he leads the Atenea Research Group. His current research interests include model-based software engineering, open distributed processing, and software quality. More information about his publications, research projects, and activities can be found at http://www.lcc.uma.es/~av, or contact him at av@lcc.uma.es.