Capturing and Querying Uncertainty in RDF Stream Processing

Robin Keskisärkkä[1], Eva Blomqvist[1], Leili Lind[1,2], and Olaf Hartig[1]

[1] Linköping University, Linköping, Sweden
robin.keskisarkka|eva.blomqvist|leili.lind|olaf.hartig@liu.se
[2] RISE Research Institutes of Sweden/Division Digital Systems, Linköping, Sweden
leili.lind@ri.se

Abstract RDF Stream Processing (RSP) has been proposed as a candidate for bringing together the Complex Event Processing (CEP) paradigm and the Semantic Web standards. In this paper, we investigate the impact of explicitly representing and processing uncertainty in RSP for the use in CEP. Additionally, we provide a representation for capturing the relevant notions of uncertainty in the RSP-QL* data model and describe query functions that can operate on this representation. The impact evaluation is based on a use-case within electronic healthcare, where we compare the query execution overhead of different uncertainty options in a prototype implementation. The experiments show that the influence on query execution performance varies greatly, but that uncertainty can have noticeable impact on query execution performance. On the other hand, the overhead grows linearly with respect to the stream rate for all uncertainty options in the evaluation, and the observed performance is sufficient for many use-cases. Extending the representation and operations to support more uncertainty options and investigating different query optimization strategies to reduce the impact on execution performance remain important areas for future research.

Keywords: RSP · CEP · Uncertainty · RSP-QL

1 Introduction

Complex Event Processing (CEP) provides techniques for continuously analyzing streaming data to detect patterns of interest. A *simple event* is used to denote anything that happens, or is contemplated as happening, while a *complex event* summarizes, represents, or denotes a set of simple events [17]. Existing CEP systems are generally not well-suited for integrating background data, supporting data interoperability, and reasoning [3]. Previous research has therefore proposed RDF Stream Processing (RSP) as a candidate for bringing together the CEP paradigm and Semantic Web standards [3,6,8,10], specifically targeting information integration and reasoning.

Representing and reasoning with uncertainty has been recognized as a critical aspect for dealing with imprecise, incomplete, and noisy data in CEP [1,4,5].

However, to the best of our knowledge, no RSP system or model exists that provides any support for representing, processing, or propagating uncertainty. Hence, little is known about how uncertainty can be incorporated into RSP and what the properties of possible approaches to do so would be. The goal of this paper is to close this gap by addressing the following overall research question.

RQ: *What is the performance impact of incorporating uncertainty into RSP?*

To address this question, we focus on different ways of explicitly managing uncertainty using probabilistic approaches. The main contribution of our paper is an evaluation of the performance impact of uncertainty in RSP processing. Additionally, we provide a representation for capturing relevant notions of uncertainty, a formal description of the query operations that can operate on the proposed representation, and a prototype implementation.

In Section 2, we describe the use-case scenario that is used in the evaluation of the prototype system. Section 3 presents background and related work, and Section 4 describes how uncertainty is represented, while Section 5 describes the query operations that can operate on this representation. Section 6 describes how the query operations are leveraged to support different uncertainty options. Section 7 presents an evaluation on the prototype implementation. Finally, Section 8 summarizes the main conclusions of this work.

2 Use-Case Scenario

This section describes a scenario originating from the recent research project E-care@home[3]. The goal of the project was to develop technical solutions to improve the care of elderly patients in their homes. The E-care@home system uses Semantic Web standards, and RSP provides a way of bridging the gap between heterogeneous background data, and the detection of complex events from streaming data.

The requirements of different stakeholders have been documented in a project deliverable, which covers a number of personas and use case scenarios based on interviews with healthcare professionals, patients and next-of-kin [16]. We use one of these scenarios as the basis for the running examples and the evaluation of performance impact. The scenario focuses on the multi-morbid persona Farida who suffers from heart failure and advanced chronic obstructive pulmonary disease (COPD):

"After her latest hospital visit a month ago the elderly, multi-morbid patient Farida was offered to be remotely monitored from the primary healthcare center [...] by use of various body and environmental sensors (the E-care@home system). Her healthcare providers also want her to assess symptoms daily and report on intake of as-needed medications. [...] During the last two weeks Farida's heart failure and COPD have been rather stable due to medication changes. [...] Now,

[3] http://ecareathome.se/

the system registers a rapidly rising body temperature and increase in heart and respiration rate. Also, her pulse is more irregular than usual." [16]

Based on the scenario above, we identify a need for the E-care@home system to be able to draw conclusions based on uncertain information, stemming from the domain itself, electronic health records (EHRs), and sensor data. For example, a patient's reported physical health parameters are affected by both the quality of the sensors, how well the sensors model a given feature, and how the sensors are used.

The changes in physical health parameters described in the scenario may point at a number of different diagnoses, such as a worsening of the heart failure, an infection, and/or COPD exacerbation (i.e., worsening of a patient's COPD condition). In the evaluation (c.f. Section 7), we focus on detecting potential COPD exacerbation events, which are often characterized by high heart rate, increased breathlessness (which leads to increased breathing rate), and low oxygen saturation levels.

3 Background and Related Work

In this section, we present the relevant background and related work. We describe the different uncertainty types that have been identified within CEP, the various ways in which uncertainty has been modeled in existing CEP systems, and briefly describe the RSP-QL* model on which we base our work.

3.1 Types of Uncertainty in CEP

Uncertainty has been recognized as a critical aspect in CEP [1,4,5], and can generally be classified based on three main types: *occurrence uncertainty*, *attribute uncertainty*, and *pattern uncertainty* [1,4,18].

Occurrence uncertainty refers to whether an event has actually occurred or not. All events in the real world are not necessarily reported, and some of the events that are reported may not have occurred at all. Occurrence uncertainty can be caused by unreliable or noisy sources, or when events are inferred from other events. For example, we may be uncertain about whether a report on a patient's heart rate should be classified as a high heart rate event.

We shall model occurrence uncertainty as a probability associated with *event type assertions*, i.e., statements that specify the type of an event. The occurrence uncertainty of an event is therefore defined with respect to a specific event type.

Attribute uncertainty refers to uncertainty in the content of events. Attributes can be incomplete, imprecise, vague, contradictory, or noisy [1,18]. Values that stem from physical phenomena always contain a degree of uncertainty that originates from, e.g., inaccuracy, imprecision, or noise in the source [5]. For example, the value reported by a heart rate sensor may be associated with a uniform distribution, representing the precision of the sensor. This means that

we may have to take into consideration the probability that the true unobserved value is within some specific interval.

We shall model attribute uncertainty using continuous probability distributions, where the distribution of a given attribute value is described by a probability density function (PDF).

Pattern uncertainty relates to uncertainty about the matching and combination of events in queries or rules. It is generally impractical, or even impossible, to list all the preconditions and consequences that should apply to a given event pattern [1]. Additionally, the causal relationships and correlations between events in a given pattern may not be certain. By supporting pattern uncertainty, the fact that we have incomplete knowledge about the system under observation can be made explicit. For example, while high heart rate is a common symptom of COPD exacerbation it can also be caused by something else entirely.

We shall model pattern uncertainty using Bayesian Networks (BNs) to encode the conditional dependencies between event types. Uncertain observations of events are supported using Pearl's method [20], where probabilistic observations are represented using *virtual nodes* added as children to the event types being observed. The conditional probability tables of the virtual nodes are defined as likelihood ratios based on the probabilities of the observed events.

The incorporation of such types of uncertainty into RSP requires two main ingredients: a representation for capturing relevant notions of uncertainty, presented in Section 4, and query operations that can operate on this representation, presented in Section 5.

3.2 Approaches to Represent Uncertainty in CEP

Probability theory provides a powerful framework for reasoning with uncertainty and is the most commonly applied framework for dealing with uncertainty within CEP [1,5]. In probability theory, statements are either true or false in some world, but the world that should be considered the correct one is uncertain. If statements are assumed to be independent, the probability of a given *world* is simply the product of the probabilities of all the statements of that world.

Cugola et al. [5] extended the rule-based event specification language TESLA to support attribute uncertainty based on this principle. The extension was created to support probability distributions as a way of expressing measurement errors, and to support uncertain matches in filters. By assuming independence between all event attributes, the probability of a detected event was calculated as the product of the uncertain matches in the query.

Automata-based CEP systems have also been extended to deal with uncertainty, but typically focus on occurrence uncertainty [1]. Generally, each rule deployed gives rise to a single automaton. An incoming event matching a sequence constraint gives rise to a state transition, and when a rule is triggered the probability of the event is calculated based on its event history.

In the approach proposed by Kawashima et al. [13], which assumes independence between all simple events, the probability of a matching rule is calculated

by summing the probabilities of all combinations that satisfy the rule. The independence assumption enables various optimization strategies that make the summation of probabilities more efficient. Other approaches, like the system proposed by Wang et al. [21], relax the assumption of independence and instead allow transitions to follow first-order Markov processes.

One of the main challenges with the automata-based approaches is the lack of an efficient mechanism to take into account background information [1]. Gillani et al. [10] partially addressed this problem in an RSP system, where the graph pattern matching was used for enriching events, and non-deterministic automata were used for temporal event pattern matching.

Probabilistic graphical models are also popular alternatives for dealing with uncertainty in CEP [1]. The two most common classes include Markov networks and BNs [5,22,23]. In both cases, the nodes of the networks represent *random variables*, and edges encode their probabilistic dependencies.

In BNs, the structure of the network encodes probabilistic dependencies, which means that domain expert knowledge can be encoded as part of the network [1]. Cugola et al. [5] implemented support for BNs in their extension of the TESLA language to support pattern uncertainty. In their approach, a BN was automatically generated for every rule deployed, with event types representing nodes in the network. Domain knowledge was then manually added to each BN by a domain expert, who would then modify and enrich the resulting network.

3.3 RSP-QL*

Several RSP models and implementations have been proposed in the past decade, each of which have included different syntax and semantics. Based on the work of the RSP community group[4], Dell'Aglio et al. defined the first version of a common RSP query model and language, referred to as RSP-QL [7,9]. As an extension of RSP-QL, in our earlier work we proposed RSP-QL* [14] to provide an intuitive and compact way for representing and querying statement-level annotations in RDF streams by leveraging RDF* and SPARQL* [11].

RDF* and SPARQL* provide an alternative to RDF reification for annotating RDF triples with metadata and querying these annotations. In RDF*, enclosing a triple using the strings '<<' and '>>' allows the triple to be used as the subject or object in other triples. For example, the triple *:bob :knows :alice* can be annotated with the source *:wikipedia* as *<<:bob :knows :alice>> :source :wipedia*. Similarly, SPARQL* is an extension of SPARQL for querying RDF* data.

4 Capturing Uncertainty in RSP

In this paper, we use the RSP-QL* model [14] to capture the different notions of uncertainty discussed in Section 3.1. While there are several ways of representing uncertainty, in this paper we limit ourselves to attribute uncertainty represented

[4] https://www.w3.org/community/rsp/

using probability distributions, occurrence uncertainty represented as probability annotations on event type assertions, and pattern uncertainty captured using BNs. The usage of these uncertainty types are covered further in the context of extensions for querying in Section 5.

In this section, we describe the three uncertainty types in more detail and introduce a way of representing uncertainty in RSP-QL*. For brevity, we omit URI prefix declarations for the remainder of the paper. We use the prefixes rspu, ecare, and sosa to refer to https://w3id.org/rsp/rspu#, http://example.org/ecare#, and http://www.w3.org/ns/sosa/, respectively.

4.1 Attribute Uncertainty as Probability Distributions

There is no standardized approach to represent probability distributions in RDF. Here we introduce an approach that uses RDF literals to represent attribute uncertainty. To this end, we define a new literal datatype, denoted by the URI *rspu:distribution*. The lexical space defining the set of valid literals for this datatype consists of all strings of the form $f(p_1, p_2, ..., p_n)$, where f is an identifier for a probability distribution type, and every p_i is a floating point number that represents a parameter value. Two such literals are considered equivalent only if they have the same identifier f and the same parameter values.

We do not specify how declarations of new probability distribution types are expressed. However, we assume that every probability distribution type specifies a probability density function (PDF), and a description of the list of parameters required by the distribution.

We here consider the normal and uniform distributions. For instance, the literal `"Normal(85, 10)"` would represent a normal distribution with a mean μ of 85, and a variance σ^2 of 10, whereas `"Uniform(30, 40)"` would represent a uniform distribution between 30 and 40.

We also provide an option for values to be annotated with probability distributions as meta data, by leveraging RDF* for statement-level annotations. We introduce the property *rspu:error* to annotate triples with measurement errors. Listing 1.1 shows an example where uncertainty is either reported directly as part of a literal, or as an annotation on a triple. A more fine-grained modeling approach for statistical distributions could potentially impact querying performance, and we consider this as part of future work.

```
1 <e1> sosa:hasSimpleResult "Normal(85,10)"^^rspu:distribution .
2 << <e1> sosa:hasSimpleResult 85 >> rspu:error "Normal(0,10)"^^rspu:distribution .
```

Listing 1.1. Attribute uncertainty represented as a literal (line 1) and as an annotation on a triple (line 2).

4.2 Occurrence Uncertainty as Probabilities

We leverage RDF* to provide a compact representation for annotating event type assertions with occurrence uncertainty. The property *rspu:hasProbability* is used

to annotate an event type assertion with a probability. Listing 1.2 illustrates an example of an event type assertion annotated with an occurrence probability of 0.95.

```
<e1> a ecare:HeartRate .
<< <e1> a ecare:HighHeartRate >> rspu:hasProbability 0.95 .
```

Listing 1.2. Occurrence probability as an annotation on an event type assertion.

An event type assertion that is not explicitly annotated with an occurrence probability is assumed to be certain, i.e., have a probability of 1. Intuitively, annotating an event with a probability of 0 would be equivalent to being certain that the event has not occurred (i.e., an explicit negation).

4.3 Pattern Uncertainty as Bayesian Networks

We identify BNs using URIs and each node in such a BN corresponds to a binary random variable, where the set of possible outcomes is limited to being true or false. An event type may be a node in a such a BN.

Adding evidence to a BN is equivalent to assigning a specific outcome to one of its nodes. However, in the presence of uncertain evidence, rather than simply assigning a state to a node, we use Pearl's method [20] to incorporate uncertain evidence by adding virtual nodes. The probability associated with the uncertain evidence is used to define the conditional probability table of the virtual node, which is expressed as a likelihood ratio in relation to the variable being observed.

For example, in Figure 1 we represent an observation of a high heart rate as a virtual node that depends on high heart rate. The conditional probability table of this virtual node represents the uncertainty of our evidence, which then indirectly affects the probability of COPD exacerbation.

Figure 1. Example illustrating how an observation of high heart rate is added as a virtual node (dashed line) that depends on the high heart rate variable.

5 Extensions for Querying

We now introduce functions for RSP-QL* queries that can operate on the representation of uncertainty described in the previous section. RSP-QL, which is based on SPARQL, is extensible and URIs can be introduced to represent custom functions in the query processor. The SPARQL specification provides no guidelines for how new functions should be defined, shared, or registered, and many RDF stores therefore provide their own methods for adding user-defined

extensions (e.g., ARQ[5], RDF4J[6], and Virtuoso[7]). Our functions for dealing with uncertainty are captured by such custom functions to support operations on both probability distributions and BNs.

5.1 Operating on Probability Distributions

We define operations over probability distributions represented as literals of the aforementioned datatype *rspu:distribution* (see Section 4.1). For these definitions, let L_{dist} be the set of all *rspu:distribution* literals and *val* be a function that maps every $x \in L_{dist}$ to the probability distribution represented by x.

Definition 1. *The URI* ***rspu:add*** *denotes the function that, for every $x \in L_{dist}$ and $k \in \mathbb{R}$, returns a literal $z \in L_{dist}$ such that $val(z)$ is the probability distribution obtained by adding the constant k to the probability distribution $val(x)$.*

Definition 2. *The URI* ***rspu:subtract*** *denotes the function that, for every $x \in L_{dist}$ and $k \in \mathbb{R}$, returns a literal $z \in L_{dist}$ such that $val(z)$ is the probability distribution obtained by subtracting k from the probability distribution $val(x)$.*

Definition 3. *The URI* ***rspu:greaterThan*** *denotes the function that, for every $x \in L_{dist}$ and $a \in \mathbb{R}$, returns a floating point number such that this number is the probability that a random sample from $val(x)$ is greater than a.*

Definition 4. *The URI* ***rspu:lessThan*** *denotes the function that, for every $x \in L_{dist}$ and $b \in \mathbb{R}$, returns a floating point number such that this number is the probability that a random sample from $val(x)$ is less than b.*

Definition 5. *The URI* ***rspu:between*** *denotes the function that, for every $x \in L_{dist}$, $a \in \mathbb{R}$, and $b \in \mathbb{R}$, returns a floating point number that is the probability that a random sample from $val(x)$ is greater than a and less than b.*

The functions listed above support some of the most common operations on probability distributions. Examples of how these functions can be used in RSP-QL* queries follow in Section 6.

5.2 Operating on Bayesian Networks

We define operations for performing Bayesian inference and while supporting uncertain evidence in BNs (see Section 4.3). We assume a collection of BNs that are available to be invoked by the query processor, where each such BN is identified by a unique URI. Each node in such a BN represents a boolean variable that is also identified by a unique URI.

An *evidence pair* for such a BN is a pair $(id, value)$ where id is a URI that denotes a node in the BN and $value \in [0, 1]$. If $value = 0$ or $value = 1$, the

[5] https://jena.apache.org/documentation/query/writing_functions.html
[6] https://rdf4j.eclipse.org/documentation/custom-sparql-functions/
[7] http://vos.openlinksw.com/owiki/wiki/VOS/VirtTipsAndTricksGuideCustomSPARQLExtensionFunction

observed event is interpreted as an observation of the state of the node identified by *id*, otherwise the value is interpreted as an observation on a virtual node with *id* as the parent.

Definition 6. *The URI* ***rspu:belief*** *denotes a function that takes as input a URI bn denoting a BN, a URI θ denoting a node in that BN, a possible outcome $o \in \{true, false\}$, and an optional list of evidence pairs for the BN. The function returns a literal with a floating point number such that this number is the probability that the outcome of the node θ is o, where this probability is inferred by from the BN with the given evidence pairs.*

Definition 7. *The URI* ***rspu:mle*** *(maximum likelihood estimation) denotes a function that takes as input a URI bn denoting a BN, a URI θ denoting a node in that BN, and an optional list of evidence pairs for the BN. The function returns a boolean literal that represents the most likely outcome of the node θ, where the outcome is inferred from the BN with the given evidence pairs*

Definition 8. *The URI* ***rspu:map*** *(maximum a posteriori probability) denotes a function that takes as input a URI bn denoting a BN, a URI θ denoting a node in that BN, and an optional list of evidence pairs for the BN. The function returns a boolean literal that represents the most likely outcome of the node θ, where the outcome is inferred from the BN with the given evidence pairs and weighted by the prior probability of θ.*

Now that we have defined the necessary operations for dealing with uncertainty, we are ready to apply them to support different uncertainty approaches.

6 Implementation of Uncertainty Approaches

The types of uncertainty introduced in Section 3.1, and the uncertainty operations described in the previous sections, can be combined with one another in different ways. We focus on a subset of the options for such a combination. That is, we consider options with each uncertainty type in isolation (Options 1–3 below), as well as a combination of pattern uncertainty with either occurrence uncertainty (Option 4) or attribute uncertainty (Option 5). We illustrate how these options are implemented using the extensions for querying from Section 5. To this end, we use the scenario outlined in Section 2, with the goal of detecting potential COPD exacerbation events based on heart rates, breathing rates, and oxygen saturation levels. For brevity, only the first example is written using RSP-QL*, while the remaining have been simplified and are expressed using SPARQL* (see the project repository for full examples).

Option 1. Attribute uncertainty can be used to support uncertain matches based on event attributes. For example, rather than specifying a *hard* limit above which a heart rate should be considered high, we can define *soft* limits and determine the probability that the limit is exceeded. Under the assumption that attributes are independent, the probability of a query result is the product of all uncertain matches in the query. The option is illustrated in Listing 1.3.

```
REGISTER STREAM <stream/copd_exacerbation> COMPUTED EVERY PT1S AS
SELECT (?p1*?p2*?p3 AS ?confidence)
FROM NAMED WINDOW <w1> ON <stream/hr> [RANGE PT1S STEP PT1S]
FROM NAMED WINDOW <w2> ON <stream/br> [RANGE PT1S STEP PT1S]
FROM NAMED WINDOW <w3> ON <stream/ox> [RANGE PT1S STEP PT1S]
WHERE {
    WINDOW <w1> {
        GRAPH ?g1 { ?e1 a ecare:HeartRate ; sosa:hasSimpleResult ?hr . }
    }
    WINDOW <w2> {
        GRAPH ?g2 { ?e2 a ecare:BreathingRate ; sosa:hasSimpleResult ?br . }
    }
    WINDOW <w3> {
        GRAPH ?g3 { ?e3 a ecare:OxygenSaturation ; sosa:hasSimpleResult ?ox . }
    }
    BIND( rspu:greaterThan(?hr, 100) AS ?p1 )
    BIND( rspu:greaterThan(?br, 30) AS ?p2 )
    BIND( rspu:lessThan(?ox, 90) AS ?p3 )
}
```

Listing 1.3. Query illustrating attribute uncertainty (option 1), where attributes are reported as rspu:distribution literals.

Option 2. Occurrence uncertainty can be used to calculate the probability of a result based on the events captured. Under the assumption that events are independent, the probability of a given result is the product of the event type assertion probabilities of all matched events. The option is illustrated in Listing 1.4.

```
SELECT ( ?p1*?p2*?p3 AS ?confidence )
WHERE {
    << ?e1 a ecare:HighHeartRate >> rspu:hasProbability ?p1 .
    << ?e2 a ecare:HighBreathingRate >> rspu:hasProbability ?p2 .
    << ?e3 a ecare:LowOxygenSaturation >> rspu:hasProbability ?p3 .
}
```

Listing 1.4. Query illustrating occurrence uncertainty (option 2).

Option 3. Pattern uncertainty is captured by conditional dependencies between events, where the dependencies are represented in an underlying BN. We assume that events are certain by rounding off probabilities to the nearest integer (i.e., observations are simplified to true or false). The probability of a complex event is found from the BN after setting the observations of the captured events. The option is illustrated in Listing 1.5.

```
SELECT ?confidence
WHERE {
    << ?e1 a ecare:HighHeartRate >> rspu:hasProbability ?p1 .
    << ?e2 a ecare:HighBreathingRate >> rspu:hasProbability ?p2 .
    << ?e3 a ecare:LowOxygenSaturation >> rspu:hasProbability ?p3 .
    BIND(rspu:belief(ecare:COPDExacerbation, true,
            ecare:HighHeartRate, IF(?p1 > .5, 1, 0),
            ecare:HighBreathingRate, IF(?p2 > .5, 1, 0),
            ecare:LowOxygenSaturation, IF(?p3 > .5, 1, 0)) AS ?confidence)
}
```

Listing 1.5. Query illustrating pattern uncertainty (option 3).

Option 4. Pattern uncertainty can be combined with occurrence uncertainty to support uncertain evidence. Unlike the approach mentioned above (option 3), which assumes that all matched events are certain, the occurrence uncertainty of each matched event is used as part of the evidence. The option is illustrated in Listing 1.6.

```
SELECT ?confidence
WHERE {
    << ?e1 a ecare:HighHeartRate >> rspu:hasProbability ?p1 .
    << ?e2 a ecare:HighBreathingRate >> rspu:hasProbability ?p2 .
    << ?e3 a ecare:LowOxygenSaturation >> rspu:hasProbability ?p3 .
    BIND(rspu:belief(ecare:COPDExacerbation, true,
            ecare:HighHeartRate, ?p1,
            ecare:HighBreathingRate, ?p2,
            ecare:LowOxygenSaturation, ?p3) AS ?confidence)
}
```

Listing 1.6. Query illustrating pattern uncertainty combined with occurrence uncertainty (option 4).

Option 5. Pattern uncertainty can also be combined with attribute uncertainty, where the probability of an event type is derived as part of the query itself based on uncertain matches on event attributes. The option is illustrated in Listing 1.7.

```
SELECT ?confidence
WHERE {
    ?e1 a ecare:HeartRateEvent ; sosa:hasSimpleResult ?hr .
    ?e2 a ecare:BreathingRateEvent ; sosa:hasSimpleResult ?br .
    ?e3 a ecare:OxygenSaturationEvent ; sosa:hasSimpleResult ?ox .
    BIND(rspu:belief(ecare:COPDExacerbation, true,
            ecare:HighHeartRate, rspu:greaterThan(?hr, 100),
            ecare:HighBreathingRate, rspu:greaterThan(?br, 30),
            ecare:LowOxygenSaturation, rspu:lessThan(?ox, 90)) AS ?confidence)
}
```

Listing 1.7. Query illustrating pattern uncertainty combined with attribute uncertainty (option 5).

7 Impact Evaluation

Now we are ready to study the impact that integrating uncertainty in RSP may have in terms of query execution performance. We first validate the output of the prototype implementation with respect to the different uncertainty options described in the previous section. We then evaluate the performance overhead of these uncertainty options with respect to different input stream rates.

7.1 Experiment Setup

For the evaluation, we introduce a *test driver* responsible for generating event streams. The test driver generates the complex events to be detected (i.e., the ground truth) and the set of low-level events from which these complex events are to be detected. The test driver allows the stream rate and the uncertainty in the generated streams to be varied.

The experiments are based on the use-case described in Section 2, with the goal of detecting potential COPD exacerbation events based on heart rates, breathing rates, and oxygen saturation levels. The thresholds considered for these parameters are typically patient and context specific. For the experiments, we simplify the scenario by fixing these bounds as follows. Heart rates are considered high if they exceed 100, breathing rates are considered high if they exceed 30, and oxygen saturation levels are considered low if they are below 90. The streams generated by the driver correspond to the types of preprocessed sensor streams that would be expected in the E-care@home system.

The test driver generates events based on the BN shown in Figure 2. The event generation starts from the complex event that is to be detected. A COPD exacerbation event is created and assigned a truth value sampled from a Bernoulli distribution based on its prior probability. The truth value of the event constitutes the ground truth. For each COPD exacerbation event, the test driver then generates three reference events: a high heart rate event, a high breathing rate event, and a low oxygen saturation event. Each of these events are assigned truth values sampled from Bernoulli distributions based on their conditional probabilities, given the state of the COPD exacerbation event.

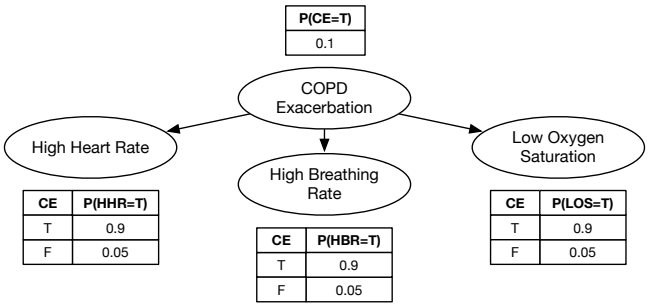

Figure 2. The BN used by the test driver in the generation of event streams.

The test driver then samples attribute values for each event based on a *degree of attribute uncertainty*. The degree of attribute uncertainty is defined as the probability that a random sample does not match the expected threshold value. For example, for a true high heart rate event and an attribute uncertainty of 10%, the value is sampled from a normal distribution such that there is a 10% probability that the sample is below 100.

For occurrence uncertainty, the test driver assigns new truth values to the events. The *degree of occurrence uncertainty* is defined as the probability that a reported event type assertion does not match the reference truth value. For example, for a true high heart rate event and an occurrence uncertainty of 5%, the truth value is (re-)sampled such that there is a 5% probability that the event will be reported as false, and the event type assertion is annotated with

a probability of 95%. The queries used in the evaluation have been excluded for brevity but are available in the project repository[8].

We base the experiments on the RSPQLStarEngine[9], which is a prototype system implementing the RSP-QL* model [14]. The query engine was extended to support SPARQL value functions along with the extensions described in Section 5. Operations on probability distributions were implemented using the library Commons Math[10], which provides support for operations on the most common probability distributions. Support for Bayesian inference was provided by using the library jSMILE (v.1.4.0) developed by BayesFusion[11]. All experiments were performed on a MacBook Pro 2015, with a quad-core 2.8 GHz Intel Core i7 processor, 16 GB of 1600MHz DDR3, and 8 GB of memory allocated to the JVM. The prototype, along with the experiment files and queries, is available under the MIT License.

7.2 Validation

The validation of the prototype implementation follows a design similar to that of Moreno et al. [19] and Cugola et al. [5]. The original set of COPD exacerbation events generated by the test driver is used as the basis for an *oracle* that can be used to check if an event occurred within a given time interval.

Due to uncertainty, a large number of detected events will have small but non-zero probabilities. The probability associated with a detected event can be interpreted as the confidence we have in it. In the validation, we apply a *confidence threshold* below which detected events are assumed to be false. Regardless of the uncertainty option used, both the true positive rate (i.e., the ratio of correctly detected events) and the false positive rate (i.e., the ratio of wrongly detected events) is expected to go down when the threshold increases. Higher degrees of uncertainty are expected to reduce the overall area under the resulting ROC curve.

For the validation, the stream rate was fixed at 1,000 events/sec per event stream, and the confidence thresholds were varied between 0 and 0.99. We illustrate the observed trade-off between true positives and false positives for two of the uncertainty options in Figure 3, where each point on a line represents a specific confidence threshold for accepting a detected event. The top left corner of the ROC box represents a perfect system. The trade-off between true positives and false positives follows the expected pattern for all uncertainty options and degrees of uncertainty.

[8] https://www.w3id.org/rsp/rspu
[9] https://github.com/keski/RSPQLStarEngine
[10] https://commons.apache.org/proper/commons-math/ (version 3.6.1)
[11] https://www.bayesfusion.com/

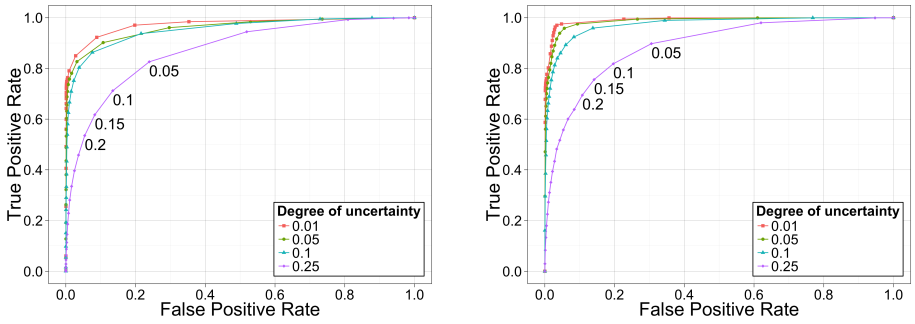

Figure 3. ROC curves for option 1 (left) and option 5 (right), with labels showing a subset of the confidence threshold values.

7.3 Performance Impact

We evaluate the impact of the different uncertainty options in terms of query execution times. The queries used for the evaluation are similar to those used for the validation in the previous section, but do not check the results against the oracle. Notably, the degree of uncertainty does not impact the overall query complexity, since the confidence for each result needs to be assessed for every potential COPD exacerbation event. For the experiments, we therefore fix the degree of uncertainty at 0.05.

In the experiments, we vary the stream rates between 100 and 3,000 events/sec per input stream, which corresponds to a total of 300–9000 events/sec (equivalent to 2,100–63,000 quads/sec). These stream rates are similar to those previously used in the testing of RSP systems [2,15]. Table 1 shows the average query execution times for different stream rates, and the relative overhead compared to a baseline query in which uncertainty is ignored completely.

Table 1. Average query execution times (ms) for each of the uncertainty options with respect to stream rates (events/second per stream). The added overhead relative to the baseline query (where uncertainty is ignored) is given in parentheses.

Rate	Option 1	Option 2	Option 3	Option 4	Option 5
100	3.07 (2.73)	1.11 (0.35)	4.37 (4.32)	6.48 (6.88)	9.97 (11.13)
500	12.69 (3.20)	4.82 (0.60)	18.23 (5.04)	25.59 (7.48)	38.88 (11.88)
1000	25.84 (2.49)	10.32 (0.39)	36.07 (3.87)	52.81 (6.12)	82.89 (10.18)
1500	37.99 (2.25)	15.22 (0.30)	55.04 (3.70)	80.69 (5.90)	123.59 (9.56)
2000	51.65 (2.79)	19.70 (0.45)	74.44 (4.47)	110.72 (7.13)	166.03 (11.20)
2500	69.22 (3.07)	26.85 (0.58)	98.95 (4.82)	146.04 (7.59)	208.74 (11.28)
3000	86.80 (2.93)	35.08 (0.59)	130.55 (4.91)	194.53 (7.81)	286.89 (11.99)

Option 2 only adds an overhead of around 50%, whereas option 1 and option 3 increase overall execution time by a factor of 3 and 4 respectively. Option 4 increases query execution time by a factor of 7, while option 5 adds an overhead that is in the order of one magnitude.

The stream rates that can realistically be supported are difficult to generalize, due to aspects such as query and data complexity, number of parallel streams, and technical implementation details. However, the results show that the average query execution time increases linearly as a function of the stream rate, and that the cost of the uncertainty operations remain constant. This means that while the impact of explicitly considering uncertainty can have noticeable impact on query execution performance, the evaluation suggests that the performance is still sufficient for many use-cases.

8 Conclusions

To the best of our knowledge, this paper presents the first work on investigating the impact of incorporating uncertainty in the RSP context. We have defined s representation for capturing relevant notions of uncertainty in RSP, provided a formal description of the query operations that can operate on this representation, and evaluated the impact on query execution performance in a prototype implementation.

The cost of the different uncertainty options was shown to vary greatly, but the results show that continuous query execution could be supported at realistic stream rates even for the computationally expensive uncertainty options. Explicitly managing uncertainty in RSP also provides a lot of flexibility, since the user can choose when and where a given uncertainty option should be applied.

The representation of uncertainty used does not require any modifications to the underlying RSP-QL* query semantics. This means that the extensions could be supported in any engine that supports RSP-QL, since RDF* and SPARQL* can be viewed simply as syntactic sugar on top of RDF and SPARQL [12].

Extending the representation to support additional uncertainty options, such as fuzzy logic, and investigating query optimization strategies to reduce the impact on execution performance remain important areas for future research.

References

1. Alevizos, E., Skarlatidis, A., Artikis, A., Paliouras, G.: Probabilistic Complex Event Recognition: A Survey. ACM Computing Surveys **50** (2017)
2. Ali, M.I., Ono, N., Kaysar, M., Shamszaman, Z., Pham, T.L., Gao, F., Griffin, K., Mileo, A.: Real-Time Data Analytics and Event Detection for IoT-Enabled Communication Systems. SSRN Electronic Journal (2017)
3. Anicic, D., Rudolph, S., Fodor, P., Stojanovic, N.: Stream Reasoning and Complex Event Processing in ETALIS. Semantic Web Journal **3**(4), 397–407 (2012)
4. Artikis, A., Etzion, O., Feldman, Z., Fournier, F.: Event Processing Under Uncertainty. In: Proc. of the 6th ACM Int. Conf. on Distr. Event-Based Systems (2012)

5. Cugola, G., Margara, A., Matteucci, M., Tamburrelli, G.: Introducing Uncertainty in Complex Event Processing: Model, Implementation, and Validation. Computing **97**(2), 103–144 (2015)
6. Dao-Tran, M., Le-Phuoc, D.: Towards Enriching CQELS with Complex Event Processing and Path Navigation. In: Proc. of the 1st Workshop on High-Level Declarative Stream Processing (2015)
7. Dell'Aglio, D., Calbimonte, J.P., Della Valle, E., Corcho, O.: Towards a Unified Language for RDF Stream Query Processing. In: Revised Selected Papers of the ESWC 2015 Satellite Events on The Semantic Web (2015)
8. Dell'Aglio, D., Dao-Tran, M., Calbimonte, J.P., Le Phuoc, D., Della Valle, E.: A Query Model to Capture Event Pattern Matching in RDF Stream Processing Query Languages. In: Knowledge Engineering and Knowledge Management (2016)
9. Dell'Aglio, D., Della Valle, E., Calbimonte, J.P., Corcho, O.: RSP-QL Semantics: a Unifying Query Model to Explain Heterogeneity of RDF Stream Processing Systems. Int. Journal on Semantic Web and Information Systems **10**(4), 17–44 (2014)
10. Gillani, S., Zimmermann, A., Picard, G., Laforest, F.: A Query Language for Semantic Complex Event Processing: Syntax, Semantics and Implementation. Semantic Web Journal **10**, 53–93 (2019)
11. Hartig, O.: Foundations of RDF* and SPARQL* – An Alternative Approach to Statement-Level Metadata in RDF. In: Proc. of the 11th AMW Workshop (2017)
12. Hartig, O., Thompson, B.: Foundations of an Alternative Approach to Reification in RDF. CoRR **abs/1406.3399** (2014)
13. Kawashima, H., Kitagawa, H., Li, X.: Complex Event Processing over Uncertain Data Streams. In: Proc. of 3PGCIC (2010)
14. Keskisärkkä, R., Blomqvist, E., Lind, L., Hartig, O.: RSP-QL*: Enabling Statement-Level Annotations in RDF Streams. In: Proc. of SEMANTiCS (2019)
15. Le-Phuoc, D., Dao-Tran, M., Pham, M.D., Boncz, P., Eiter, T., Fink, M.: Linked Stream Data Processing Engines: Facts and Figures. In: The Semantic Web – ISWC 2012, vol. 7650, pp. 300–312. Springer, Berlin, Heidelberg (2012)
16. Lind, L., Prytz, E., Lindén, M., Kristoffersson, A.: Use cases unified description. E-care@home project Milestone Report MSR5.1b (Project Internal) (2017)
17. Luckham, D., Schulte, R.: Event Processing Glossary Version 2.0. Event Processing Society (2011)
18. Margara, A., Urbani, J., van Harmelen, F., Bal, H.: Streaming the Web: Reasoning over Dynamic Data. Journal of Web Semantics **25**, 24–44 (2014)
19. Moreno, N., Bertoa, M., Burgueno, L., Vallecillo, A.: Managing Measurement and Occurrence Uncertainty in Complex Event Processing Systems. IEEE Access **7**, 88026–88048 (2019)
20. Pearl, J.: Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference. Morgan Kaufmann Publishers, San Francisco, California (1988)
21. Wang, Y.H., Cao, K., Zhang, X.M.: Complex Event Processing over Distributed Probabilistic Event Streams. Comput. Math. Appl. **66**(10), 1808–1821 (2013)
22. Wasserkrug, S., Gal, A., Etzion, O.: A Model for Reasoning with Uncertain Rules in Event Composition Systems. In: 21st Conf. on Uncertainty in Artificial Int. (2005)
23. Wasserkrug, S., Gal, A., Etzion, O., Turchin, Y.: Complex Event Processing over Uncertain Data. In: 2nd Int. Conf. on Distr. Event-based Systems (2008)