SENSdesc: Connect Sensor queries and Context

Dörthe Arndt[1], Pieter Bonte[2], Alexander Dejonghe[2], Ruben Verborgh[1], Filip De Turck[2] and Femke Ongenae[2]
[1]*IDLab, Department of Electronics and Information Systems, Ghent University – imec*
[2]*IDLab, Department of Information Technology, Ghent University – imec*
doerthe.arndt@ugent.be

Keywords: N3, Stream Reasoning, Rule-Based Reasoning, Proofs, IoT

Abstract: Modern developments confront us with an ever increasing amount of streaming data: different sensors in environments like hospitals or factories communicate their measurements to other applications. Having this data at disposal faces us with a new challenge: the data needs to be integrated to existing frameworks. As the availability of sensors can rapidly change, these need to be flexible enough to easily incorporate new systems without having to be explicitly configured. Semantic Web applications offer a solution for that enabling computers to 'understand' data. But for them the pure amount of data and different possible queries which can be performed on it can form an obstacle. This paper tackles this problem: we present a formalism to describe stream queries in the ontology context in which they might become relevant. These descriptions enable us to automatically decide based on the actual setting and the problem to be solved which and how sensors should be monitored further. This helps us to limit the streaming data taken into account for reasoning tasks and make stream reasoning more performant. We illustrate our approach on a health-care use case where different sensors are used to measure data on patients and their surrounding in a hospital.

1 Introduction

With the development of new technologies and the possibility to rapidly produce and store more and many different kinds of data, health care systems have new opportunities: Next to the classical electronic health records (EHR) of patients, they can use background data clarifying diseases, their diagnosis and possible cure, organisational data (e.g. the availability and competences of staff members in a hospital) but also data produced by sensors for example to measure values on patients (e.g. their blood pressure or pulse), or to control the patient's ambience in a hospital (e.g. the illumination or temperature in his room). To combine these in a meaningful way, Semantic Web reasoning systems are particularly well suited: their declarative approach allows the user to explain concepts represented by the data and to specify how for example the known EHR and newly measured values of a patient relate to each other. If the system knows how to interpret the values of a specified kind of sensor, it is easy to add and remove new instances of this type. But as most computer systems, Semantic Web implementations also have to cope with the challenges of Big data[1]: if they have to deal with with a huge *volume* and different forms of data (*variety*), which can come in a high *velocity*, for example in data streams, and in different levels of quality (*veracity*), the performance of the systems can be rather poor.

In this paper we aim to tackle this problem: in a setting where many sensors are present and a request constantly needs to be answered, our approach finds the sensors relevant for this specific goal. These sensors and their results can then be monitored more closely while others, not relevant for the problem, can be ignored. By that, the amount of data taken into account can be reduced. To do so, we developed a special format to describe possible sensor queries consisting of three parts: the context in which a sensor query becomes relevant, the query itself, and the consequence in terms of the ontology used. Together with the related knowledge the descriptions enable a reasoner to produce a formal proof which we use to find the sensor queries contributing to the goal.

The remainder of this paper is structured as follows: In Section 2 we illustrate a scenario in which our application can be used. This serves as a running

[1] http://www.ibmbigdatahub.com/infographic/four-vs-big-data

example throughout the whole paper. In Section 3 we provide the background knowledge needed to understand our solution. After that, in Section 4 we explain the details of our implementation including our new description format and the use of formal proofs to determine relevant sensor queries. In Section 5 we discuss the relation of our approach to other research. We conclude our paper by Section 6.

2 Use case

The use case of this paper is set in a hospital. This hospital is equipped with a large number of different sensors to monitor, among others, the temperature and light intensity in a room, values related to the patient's health, e.g. pulse, blood pressure and body temperature, and the location of staff members and patients. We want to enable a computer to use this data in order to answer requests or to detect critical situations. These situations do not only depend on the sensor values themselves, but also on their surrounding: patients, their diseases, settings in the hospital like rooms and locations of the sensors and many more aspects could influence the decision, whether a sensor value is normal or alarming. Such surroundings can change: patients enter and leave the hospital, sensors are added or removed. Our system should thus take the context into account, and easily adapt to possible changes. We therefore use Semantic Web technology and describe the data itself and the context of the sensor setting in a machine understandable way. If the system *understands* the concept of a light sensor, it knows how to deal with a new instance of such a device. Only the new sensor and its surrounding need to be described, the concrete use of the sensor, e.g. threshold values, can be deducted from the knowledge. The ontology we use includes the hospital and its rooms, the patients and their diseases, and the descriptions and locations of the different sensors. We aim to find in the current setting all possible kinds of problems which could be detected by a sensor.

In our ontology, we call these problems *faults*. Such faults could be caused by many reasons like an alarming sensor value (e.g. the luminance, sound or temperature in a room is a above or below a critical threshold), or faulty sensors (e.g. a temperature, luminance or sound sensor does not work properly). As stated, faults depend on context like patient profiles and locations or the general sensor set-up in the hospital: While a pulse of 110/min might be alarming for a grown-up, it is totally normal for an infant. Similarly, whether the light in a room is too bright depends on who is using the room: a patient suffering from a concussion is more sensitive to light than a patient treated for Parkinson. In an empty room the light intensity does not influence the well-being of the patients and is therefore irrelevant in our case.

This relevance of context makes the detection of faults a complex task. While in a small or simple setting where either context is irrelevant or the number of sensors is small, it is no problem to perform reasoning on the sensor data whenever a value changes, this is different for the given situation. Each reasoning task takes time (depending on its complexity the task can take seconds on a common computer (Arndt et al., 2015)) and the performance of computers present in hospitals is often limited.

We therefore want to find out early in the current setting (e.g. rooms of patients, patients' diagnosis):

1. Which sensors are relevant to our request, i.e. which sensors could detect a fault?
2. Which values are critical for these sensors?

This allows us to monitor only these sensors and queries, and just perform complex reasoning when critical values are measured.

3 Background

This section introduces the background required to understand the details of our implementation. In particular these are: the logic employed, N3 (Berners-Lee et al., 2008), the notion of proofs in N3, and the possibility of performing OWL reasoning in N3.

3.1 N3Logic

Invented about a decade ago by Tim Berners-Lee et al. (Berners-Lee et al., 2008), N3Logic (N3) forms a superset of RDF (Cyganiak et al., 2014). As in RDF, simple N3 statements are expressed by triples consisting of URIs, literals and blank nodes (starting with `_:`). The latter stand for existentially quantified variables. The formula

```
_:x :likes :ice-cream.
```

means: *'There exists someone who likes ice cream.'* Additionally to these existential variables, N3 supports universal variables starting with the symbol `?`:

```
?x :likes :ice-cream.
```

means: *'Everyone likes ice-cream.'* Universal variables are mostly used in rules, which we state using curly brackets `{ }` and the implication symbol `=>`:

```
{?x a :Child}=>{?x :likes :ice-cream}.   (1)
```

```
1 PREFIX : <http://example.org/sensdesc#>
2 PREFIX r:
  <http://www.w3.org/2000/10/swap/reason#>
3
4 <#lemma3> a r:Inference;
5  r:gives {:Ben :likes :ice-cream.};
6  r:evidence (<#lemma2>);
7  r:rule <#lemma1>.
```

Listing 1: Example inference step. Formula 3 gets derived by applying the rule in Formula 1 (lemma 1) on Formula 2 (lemma 2).

stands for: *'If x is a child then x likes ice cream.'* or shorter *'All children like ice-cream.'* The same notation of curly brackets can be used to cite formulas:

```
:Ana :knows {:Ben :likes :ice-cream.}.
```

This means: *'Ana knows that Ben likes ice-cream.'* A more detailed introduction to the syntax and semantics of N3 can be found in the corresponding specification (Berners-Lee and Connolly, 2011).

3.2 Proofs in N3

N3 expressions as introduced above can be given to a reasoner which then derives new knowledge from that and provides a proof. A proof is a description of all the steps performed to come to a conclusion. Given a set of triples and rules *D*, and a goal *g*, i.e. a formula which should be verified, these steps can be: (1) reading a valid formula, i.e. an *axiom* from *D*, this formula can either be a simple triple or rule, or a conjunction of these, (2) perform *conjunction-elimination*, i.e. select one conjunct from a conjunction, (3) apply a rule to a set of triples (*modus ponens*), or (4) perform a *conjunction introduction*, i.e. form the conjunction of two or more proven formulas.

The proofs produced by N3 reasoners are themselves given in the N3 format using the vocabulary of the Semantic Web Application Platform (SWAP) (Berners-Lee, 2000). Each step mentioned above has a name: (1) `r:Parsing`[2] is the name for axiomatizing, (2) `r:Extraction` stands for conjunction elimination, (3) `r:Inference` refers to the modus ponens, and (4) `r:Conjunction` to conjunction introduction. The connection of these steps to their results is expressed using the predicate `r:gives`. The valid formulas directly taken into account for the proof steps `r:Extraction`, `r:Conjunction` and `r:Inference` are displayed by referring the reasoning step which lead to them (`r:because`, `r:component`, and the pair

[2]The prefix `r` refers here and for the remainder of this paper to the name space `http://www.w3.org/2000/10/swap/reason#`.

```
1  PREFIX WA: <...WSNextensionAccio.owl#>
2  PREFIX SN: <...SSNiot.owl#>
3  PREFIX owl:<http://www.w3.org/2002/07/owl#>
4
5  WA:LuminanceAboveThresholdFault
6   a owl:Class ;
7   owl:equivalentClass [
8    a owl:Restriction;
9    owl:onProperty SN:hasSymptom;
10   owl:someValuesFrom
11    WA:LuminanceAboveThresholdSymptom
12  ].
```

Listing 2: Axiom using `owl:someValuesFrom`.[4]

`r:rule` and `r:evidence`, respectively). In case of `r:Parsing` the source is given (using `r:source`).

As the step `r:Inference` is particularly important for our application we consider an example: Listing 1 explains, how the reasoner[3] applies Rule 1 to the fact:

```
:Ben a :Child.                    (2)
```

and comes to the result

```
:Ben :likes :ice-cream.           (3)
```

In the example, Rule 1 is obtained by a `r:Parsing` and `r:Extraction` (not displayed here) which together lead to Lemma 1, the rule is thus the result of this lemma. The same steps performed on another input lead to Lemma 2 which has Formula 2 as a result.

Listing 1 specifies Lemma 3 which is an instance of the proof step `r:Inference` (line 4). This lemma takes the result of Lemma 2 (i.e. Formula 2) as input triple. This is indicated using the predicate `r:evidence` (line 6). It then applies the rule derived by Lemma 1 (i.e. Rule 1) on it. This relation is expressed by using the predicate `r:rule` (line 7). This leads to (`r:gives`) Formula 3 as a conclusion (line 5).

A proof contains all such applications of rules which contributed to the derivation of the goal. Further information about the proof calculus can be found in our previous work (Verborgh et al., 2017).

3.3 OWL and N3

Next to N3 and RDF there is a third format typically used in the Semantic Web to represent knowledge, the Web Ontology Language (OWL) (Bock et al., 2012). This Description-Logic-based format is very strong in its expressiveness but the tableaux algorithm normally

[3]All proofs and proof steps shown in this paper are produced by the EYE reasoner (Verborgh and De Roo, 2015).

[4]In this and all the following listings the dots (...) in the prefix declaration stand for `http://IBCNServices.github.io/Accio-Ontology/`.

```
PREFIX owl:<http://www.w3.org/2002/07/owl#>

{?D owl:equivalentClass ?C.
 ?C owl:someValuesFrom ?Y.
 ?C owl:onProperty ?P.
 ?U ?P ?V.
 ?V a ?Y.
}=>{?U a ?D}.
```

Listing 3: OWL RL rule for `owl:someValuesFrom`.[5]

```
PREFIX owl:<http://www.w3.org/2002/07/owl#>

{?x a ?C
 ?C owl:equivalentClass
   [ owl:onProperty ?property;
     owl:someValuesFrom ?from ].
}=>{?x ?property _:e. _:e a ?from.}.
```

Listing 4: Existential rule for `owl:someValuesFrom`.

used to reason on it is rather slow compared to other mechanisms (Krötzsch, 2012; Arndt et al., 2015).

Our implementation uses OWL ontologies over which we reason using the rule based logic N3. OWL has the sub-profiles (Calvanese et al., 2012), OWL RL, QL and EL. The profile OWL-RL contains all constructs expressible by simple Datalog (Abiteboul et al., 1995) rules. This is natively supported by N3 (Arndt et al., 2016). But as N3 also covers existential rules, i.e. rules with existentially quantified variables in their conclusion, it also supports OWL constructs not present in OWL-RL. An example is the OWL predicate `owl:someValuesFrom` for which in OWL RL only supports the subclass version.

The property restriction `owl:someValuesFrom` for a predicate *p* on a class *C* means, that each instance of *C* has at least one value connected to it via *p*. The object of `owl:someValuesFrom` determines the class to which that connected value belongs. To illustrate that we display the description of the example class `WA:LuminanceAboveThresholdFault` in Listing 2. This class covers a special kind of faults and is defined (using `owl:equivalentClass`) as the class of all instances which have a symptom (`SN:hasSymptom`) being an instance of the class `WA:LuminanceAboveThresholdSymptom`. Given

```
:observation1 SN:hasSymptom [
 a WA:LuminanceAboveThresholdSymptom]. (4)
```

the description enables the reasoner to apply the RL rule in Listing 3 and conclude that

```
:observation1 a
    WA:LuminanceAboveThresholdFault. (5)
```

But the mechanism also needs to work the other way around. Given an instance of a class with a `owl:someValuesFrom` restriction on a property *p* like in Formula 5 where `:observation1` is a `WA:LuminanceAboveThresholdFault`, it can be derived that there *exists* an object of the class indicated in the class axiom which is connected to that instance via *p*. In our example, that is an instance of the class `WA:LumincanceAboveThresholdSymptom` connected to our `:observation1` via the property `SN:hasSymptom`. The existential rule displayed in Listing 4 produces the expected result, it derives Formula 4 from Formula 5 and Listing 2. By implementing this and other rules, we extended the set of OWL concepts supported by our rule based implementation.

4 Implementation

After having explained use case and background knowledge, we now clarify how we solve the problem at hand, the identification of sensors and sensor queries which can be used to detect alarming situations. The basic idea of our approach is, given a description of the hospital's setting, to use existential rules to describe possible sensor queries and their meaning in terms of the ontology. We call that format SENSdesc. Together with a general goal, i.e. a request to be answered, a reasoner can employ all the knowledge to produce a proof. If this proof contains the instantiated version of a sensor query, this occurrence indicates that in the given context this query is relevant and needs to be monitored. We explain the details of this idea below.

4.1 Description of context

To be able to use the knowledge about context in a hospital to determine which sensors need to be observed carefully and for which values, this context knowledge needs to be described in a machine understandable way. In our implementation we use the ACCIO ontology[6] (Ongenae et al., 2014). This OWL DL ontology makes use of other well established ontologies e.g. the Semantic Sensor Ontology (Compton M. et al., 2012), and was designed to represent different aspects of patient care in a continuous care setting. It

[5] In the actual implementation we have extra rules to handle `owl:equivalentClass`. We just added its handling to this rule to make our example self-explaining.

[6] Available at: `https://github.com/IBCNServices/Accio-Ontology/tree/gh-pages`

```
1  PREFIX : <http://example.org#>
2  PREFIX Ar:  <...RoleCompetenceAccio.owl#>
3  PREFIX Du:  <...ontologies/DUL.owl#>
4  PREFIX SN: <...SSNiot.owl#>
5  PREFIX WA: <...WSNextensionAccio.owl#>
6
7  :bob Du:hasRole [ a Ar:PatientRole];
8     Du:hasLocation :room21;
9     WA:lightTreshold 200.
10
11 :lightsensor17
12      Du:hasLocation :room21;
13      a SN:LightSensor.
```

Listing 5: Instance of a patient with a light sensivity of 200 who is located in a room with a light sensor.

covers all concepts relevant to our use case. A simple example of the use of the ontology is given in Listing 5. We see specifications about someone named `:bob` who is a patient and currently present at a location called `:room21`. In this `:room21` there is furthermore a light sensor (`:lightsensor17`). We also see that `:bob` has a light threshold value of 200 assigned. Such an assignment can either be derived by reasoning (e.g. if Bob has a concussion and all patients with a concussion have this light threshold value) or it can be set individually to a patient (e.g. by a doctor). Below, we show how our system uses this knowledge.

4.2 Sensor queries

Our concept, SENSdesc, describes possible sensor queries via existential rules of the form

```
{ pre-condition } =>{ query-description.
                      post-condition.    }
```

where the three parts are as follows:

Pre-condition: This part specifies the kind of sensor the description is valid for, the situation in which the query can be done and additional knowledge relevant for the query.

Query description: This part specifies how exactly a query has to look like. Additional parameters relevant for the query can also be added here.

Post-condition: Here we describe the consequences of the query-result.

For pre- and post-condition it is crucial to use vocabulary specified in the surrounding ontology. Only with further background knowledge about the terms and contexts used, reasoning can take place. We illustrate the concept of a SENSdesc rule in Listing 6.

The *pre-condition* (lines 9–14) describes the situation in which our description is relevant: A patient has a light threshold defined and there is a light sensor at the same location as the patient. In our example this is the case for our patient *Bob* from Listing 5.

```
1  PREFIX : <http://example.org#>
2  PREFIX Ar:  <...RoleCompetenceAccio.owl#>
3  PREFIX Du:  <...ontologies/DUL.owl#>
4  PREFIX SN: <...SSNiot.owl#>
5  PREFIX sosa:  <http://www.w3.org/ns/sosa/>
6  PREFIX WA: <...WSNextensionAccio.owl#>
7
8  {
9  #pre-condition
10 ?p Du:hasRole [ a Ar:PatientRole];
11    Du:hasLocation ?l;
12    WA:lightTreshold ?t.
13 ?s Du:hasLocation ?l;
14    a SN:LightSensor.
15 }=>{
16 #query-description
17 ?s :sensorQuery {
18   ?s :value _:v.
19   _:v :greaterThan ?t }.
20
21 #post-condition
22 ?s sosa:hasObservation _:o.
23 _:o SN:hasSymptom [ a
24  WA:LuminanceAboveThresholdSymptom].
25 }.
```

Listing 6: Example description. For a patient for whom a lightThreshold is defined, light sensors at his location can test whether the light is below this threshold.

The *query-description*, lines 16–19, states which query has to be performed to the light sensor. The example query here tests if the measured value is above the threshold. Note that sensor and threshold value are expressed by universal variables taking values from pre-condition. Both depend on the context data.

The *post-condition*, lines 21–24, describes the consequence of a successful sensor query: if the query triggers, a `WA:LuminanceAboveThresholdSymptom` is observed. In the ontology, the observation of this symptom can—depending on the further context—have several consequences. Here, via Listing 2, we can get a `WA:LuminanceAboveThresholdFault`.

4.3 Goals

As explained in Section 3.2, an N_3 reasoner is typically invoked with a *goal*. This goal is a logical rule serving as a filter. If this rule can be applied to the given input or its logical consequences, the reasoner provides a formal proof in which the goal is the last rule applied. As mentioned in Section 2, we want to know in our example, which faults could be detected by a sensor using the current data consisting of ontology, description of the context and SENSdesc rules to

describe possible sensor queries. To do so, we search for all instances of the class `SN:Fault` the reasoner can detect. The goal for this looks as follows:

```
{?x a SN:Fault}=>{?x a SN:Fault}.    (6)
```

Here, antecedence and consequence are the same. The rule is nevertheless not redundant because it is marked as a goal. The reasoner looks for cases where this rule can be applied and provides a proof for them.

4.4 Making use of proofs

Having the goal, the different SENSdesc descriptions including the one from above, the OWL ontology and the N3 rules to perform OWL reasoning as described in Section 3.3 at our disposal, we can start an N3 reasoner and produce a proof. This proof enables us to find the sensors and the concrete sensor queries relevant to our problem. We illustrate that idea on our example. Additionally to the axioms mentioned above, our ontology also contains the triple:

```
WA:LuminanceAboveThresholdFault
            rdfs:subClassOf SN:Fault.    (7)
```

This indicates that every instance of the class `WA:LuminanceAboveThresholdFault` is also an instance of the class `SN:Fault`. Using the data displayed in this paper and the goal in Formula 6 the reasoning process produces the proof in Listing 7. From the different proof steps (lemmas) composing the proof, there is one particularly interesting: Lemma 4 (lines 27–38) describes the application of our SENSdesc rule (`r:Inference`) leading to:

```
:lightsensor17 :sensorQuery {
  :lightsensor17 :value _:sk_0.
  _:sk_0 :greaterThan 200 }.
:lightsensor17 sosa:hasObservation _:sk_1.
_:sk_1 SN:hasSymptom _:sk_2.
_:sk_2 a WA:LuminanceAboveThresholdSymptom.
```

These triples contain a concrete sensor query to the sensor `:lightsensor17`: if the value of this sensor is greater than 200 we need to add the observation of a `WA:LuminanceAboveThresholdSymptom` to the knowledge. We thus know from the the proof that in our current setting this particular sensor is important and needs to be monitored with the indicated value.

This technique of looking for inference steps applying SENSdesc rules can be generalised: If we additionally had another sensor in another room where another patient with a defined light threshold value was located, this sensor with the corresponding threshold value would also appear in the proof while light sensors located in rooms with no patients or with patients without a defined luminance threshold value will not be listed. In that way we find the sensors and queries relevant to our particular problem.

5 Related work

The approach presented in this paper is influenced by other research: In previous work, we developed a format to describe RESTful API calls via existential rules in N3, RESTdesc.[7] Just as in SENSdesc, we used existential rules consisting of three parts, precondition, call-description, and post-condition to describe possible events, in this case possible API calls. To make plans towards a goal, proofs containing instantiated descriptions of API calls were employed. We explain this idea in our previous paper (Verborgh et al., 2017). In this sense this paper can be seen as an extension of RESTdesc, which was developed for API calls, to sensor queries. Both formats, RESTdesc and SENSdesc can be combined in an environment where APIs and sensors are present.

The use case required to perform OWL reasoning in a rule-based logic. The idea to do so is not new and has been implemented in several systems. Classically, rule-based systems support the OWL RL profile. Examples are RDFox (Nenov et al., 2015) and OWLIM (Bishop et al., 2011) or the implementation of OWL RL rules in Prolog (Almendros-Jiménez, 2011). But also the other profiles have been implemented using rules: Krötsch discusses the implementation of OWL EL (Krötzsch, 2011). The support for OWL QL in OWLIM is implemented using rules (Bishop and Bojanov, 2011). In our case the rules had to be in N3, their implementation is based on our previous work on OWL RL (Arndt et al., 2015; Arndt et al., 2016).

Finally, we discuss the connection between our work and Semantic Web stream processing and reasoning. A complete overview of this topic can be found in (Dell'Aglio et al., 2017). RDF stream processing systems are classically divided into two categories: Complex Event Processing systems (CEP) and Data Stream Management Systems (DSMS). While the former normally have timestamped triples and the detection of patterns within these as subject, the latter focus on the data produced in fixed time periods, so called windows. An overview of systems can be found at (Margara et al., 2014). To query on streams, several languages have been developed, e.g. C-SPARQL, CEQLS and $\text{SPARQL}_{\text{stream}}$. Recent research aims to unify them into the language RSP-QL (Dell'Aglio et al., 2015). Our approach is independent of these languages since reasoning only happens on top of query results whose consequences are defined in the SENSdesc descriptions. In that aspect it also differs from ontology based data access (OBDA) on streams such as e.g. (Bienvenu et al., 2014). OBDA takes sensor queries as input which are

[7] `http://restdesc.org/`

```
PREFIX owl:  <http://www.w3.org/2002/07/owl#>
PREFIX SN: <http://IBCNServices.github.io/Accio-OntologySSNiot.owl#>
PREFIX sosa:  <http://www.w3.org/ns/sosa/>
PREFIX WA: <http://IBCNServices.github.io/Accio-OntologyWSNextensionAccio.owl#>
PREFIX : <http://example.org#>
PREFIX r:  <http://www.w3.org/2000/10/swap/reason#>

[] a r:Proof, r:Conjunction;
  r:component <#lemma1>;
  r:gives { _:sk_1 a SN:Fault. }.

<#lemma1> a r:Inference;
  r:gives { _:sk_1 a SN:Fault. };
  r:evidence (<#lemma2>);
  r:rule <#lemma5>.

<#lemma2> a r:Inference;
  r:gives { _:sk_1 a SN:Fault.};
  r:evidence (<#lemma3> <#lemma7>);
  r:rule <#lemma8>.

<#lemma3> a r:Inference;
  r:gives { _:sk_1 a WA:LuminanceAboveThresholdFault.};
  r:evidence (<#lemma4> <#lemma6>);
  r:rule <#lemma9>.

<#lemma4> a r:Inference;
  r:gives {
    :lightsensor17 :sensorQuery {
      :lightsensor17 :value _:sk_0.
      _:sk_0 :greaterThan 200
    }.
    :lightsensor17 sosa:hasObservation _:sk_1.
    _:sk_1 SN:hasSymptom _:sk_2.
    _:sk_2 a WA:LuminanceAboveThresholdSymptom.
  };
  r:evidence (<#lemma10>);
  r:rule <#lemma11>.

<#lemma5>  a r:Extraction;
           r:because [ a r:Parsing; r:source <Formula6>].
<#lemma6>  a r:Extraction;
           r:because [ a r:Parsing; r:source <Listing2>].
<#lemma7>  a r:Extraction;
           r:because [ a r:Parsing; r:source <Formula7>].
<#lemma8>  a r:Extraction;
           r:because [ a r:Parsing; r:source <owl_subclass_rule.n3>].
<#lemma9>  a r:Extraction;
           r:because [ a r:Parsing; r:source <Listing3>].
<#lemma10> a r:Extraction;
           r:because [ a r:Parsing; r:source <Listing5>].
<#lemma11> a r:Extraction;
           r:because [ a r:Parsing; r:source <Listing6>].
```

Listing 7: (Simplified) Example proof for the goal displayed in Formula 6 taking into account the data from Listings 2–6 and an additional rule to handle `owl:subclassOf`. Lemma 4 includes an instatiated version of the SENSdesc description.

rewritten based on ontology knowledge to easier sensor queries to perform on the stream. In our approach, we do not have a sensor query on top of ontology and streams, we have a simple goal, which can not directly contain informations about streams and or windows. This kind of information can only be expressed in the query description part of our SENSdesc rules.

6 Conclusion and future work

In this paper we presented a new format to describe possible sensor queries and explained how it can be used together with formal proofs to determine in a setting where many sensors are available, which sensors and which queries are relevant to keep track of user defined goals. Once these sensors are known, they can be carefully monitored and—in case they detect the situations they are looking for—new reasoning can be triggered. This strategy saves us from having performance problems due to constant reasoning on the output of all available sensors.

In the future we plan to improve the description of sensor queries in SENSdesc. Integrating existing formats for stream querying like RSP-QL make it easier to detect and execute the sensor queries in a proof. Furthermore, we plan to test our implementation in bigger settings and contexts where more sensors are available. Only by this, we can be sure that the performance of our approach does not suffer from the inclusion of expensive concepts like existential rules.

REFERENCES

Abiteboul, S., Hull, R., and Vianu, V., editors (1995). *Foundations of Databases: The Logical Level*. Addison-Wesley.

Almendros-Jiménez, J. M. (2011). A Prolog library for OWL RL. In *Proceedings of the 4th International Workshop on Logic in Databases*, LID '11, pages 49–56. ACM.

Arndt, D., De Meester, B., Bonte, P., Schaballie, J., Bhatti, J., Dereuddre, W., Verborgh, R., Ongenae, F., De Turck, F., Van de Walle, R., and Mannens, E. (2015). Ontology reasoning using rules in an eHealth context. In *Proceedings of the 9th International RuleML Symposium*.

Arndt, D., De Meester, B., Bonte, P., Schaballie, J., Bhatti, J., Dereuddre, W., Verborgh, R., Ongenae, F., De Turck, F., Van de Walle, R., and Mannens, E. (2016). Improving OWL RL reasoning in N3 by using specialized rules. In *Ontology Engineering: 12th Int. Experiences and Directions Workshop on OWL*.

Berners-Lee, T. (2000). Semantic Web Application Platform. `http://www.w3.org/2000/10/swap/`.

Berners-Lee, T. and Connolly, D. (2011). Notation3 (N3): A readable RDF syntax. W3C Team Submission. `http://www.w3.org/TeamSubmission/n3/`.

Berners-Lee, T., Connolly, D., Kagal, L., Scharf, Y., and Hendler, J. (2008). N3Logic: A logical framework for the World Wide Web. *Theory and Practice of Logic Programming*, 8(3):249–269.

Bienvenu, M., Cate, B. T., Lutz, C., and Wolter, F. (2014). Ontology-based data access: A study through disjunctive Datalog, CSP, and MMSNP. *ACM Trans. Database Syst.*

Bishop, B. and Bojanov, S. (2011). Implementing OWL 2 RL and OWL 2 QL rule-sets for OWLIM. In *OWLED*.

Bishop, B., Kiryakov, A., Ognyanoff, D., Peikov, I., Tashev, Z., and Velkov, R. (2011). OWLIM: A family of scalable semantic repositories. *Semantic Web*, 2(1):33–42.

Bock, C., Fokoue, A., Haase, P., Hoekstra, R., Horrocks, I., Ruttenberg, A., Sattler, U., and Smith, M. (2012). Owl 2 Web Ontology Language. W3C Recommendation. `http://www.w3.org/TR/owl2-syntax/`.

Calvanese, D., Carroll, J., Di Giacomo, G., Hendler, J., Herman, I., Parsia, B., Patel-Schneider, P. F., Ruttenberg, A., Sattler, U., and Schneider, M. (2012). Owl 2 Web Ontology Language Profiles (second edition). W3C Recommendation. `www.w3.org/TR/owl2-profiles/`.

Compton M. et al. (2012). The SSN ontology of the W3C semantic sensor network incubator group. *Web Semantics: Science, Services and Agents on the World Wide Web*, 17.

Cyganiak, R., Wood, D., and Lanthaler, M. (2014). RDF 1.1: concepts and abstract syntax. `http://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/`.

Dell'Aglio, D., Calbimonte, J.-P., Della Valle, E., and Corcho, O. (2015). *Towards a Unified Language for RDF Stream Query Processing*.

Dell'Aglio, D., Valle, E. D., van Harmelen, F., and Bernstein, A. (2017). Stream reasoning: a survey and outlook: A summary of ten years of research and a vision for the next decade. *Data Science Journal*, 1.

Krötzsch, M. (2011). Efficient rule-based inferencing for OWL EL. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence*.

Krötzsch, M. (2012). OWL 2 profiles: An introduction to lightweight ontology languages. In *Proceedings of the 8th Reasoning Web International Summer School*.

Margara, A., Urbani, J., van Harmelen, F., and Bal, H. (2014). Streaming the Web: reasoning over dynamic data. *Journal of Web Semantics*.

Nenov, Y., Piro, R., Motik, B., Horrocks, I., Wu, Z., and Banerjee, J. (2015). RDFox: a highly-scalable RDF store. In *Proc. International Semantic Web Conference*.

Ongenae, F., Duysburgh, P., Sulmon, N., Verstraete, M., Bleumers, L., De Zutter, S., Verstichel, S., Ackaert, A., Jacobs, A., and De Turck, F. (2014). An ontology co-design method for the co-creation of a continuous care ontology. *Applied Ontology*, 9(1):27–64.

Verborgh, R., Arndt, D., Van Hoecke, S., De Roo, J., Mels, G., Steiner, T., and Gabarró Vallés, J. (2017). The pragmatic proof: Hypermedia API composition and execution. *Theory and Practice of Logic Programming*, 17(1):1–48.

Verborgh, R. and De Roo, J. (2015). Drawing conclusions from Linked Data on the Web. *IEEE Software*, 32(5):23–27.