AUTOMATIC (RE)CONFIGURATION OF RSP ENGINES

Mathias De Brouwer

Stream Reasoning Workshop 2019 in Linköping, Sweden – 17 April 2019
EXAMPLE USE CASE – PATIENT BOB

Diagnosis Bob: epilepsy attack

Epilepsy patients are sensitive to light
Threshold: light should not go > 250 lumen

Query 1: generate alarm if value measured by sensor A1 is > 250

Electronic Health Record of patients
Medical domain knowledge
Hospital lay-out
Care staff

RSP engine
EXAMPLE USE CASE – PATIENT BOB

Sound sensor A0
Light sensor A1
Temperature sensor A2
...
Localization
Door/window sensor
Wearable
Motion sensor

Query 1: generate alarm if value measured by sensor A1 is > 250
Query 2: generate alarm if value measured by sensor A0 is > 30

Diagnosis Bob: concussion

Concussion patients are sensitive to light & sound
Thresholds: light should not go > 170 lumen, sound should not go > 30 decibels

Electronic Health Record of patients
Medical domain knowledge
Hospital lay-out
Care staff

RSP engine

Hospital layout
GENERAL ISSUE vs. GOAL

(Changed) context

Queries need to be manually (re)configured

By performing reasoning on changed context instead of on the real-time data streams

Queries of interest are automatically derived & (re)configured
OUR SOLUTION: PROCESS

**Ontology**

Sensor Query Rule(s) (Includes Generic Query)

Goal (What to Detect)

Input data (defined once)

Context

New or changed
E.g. for a patient/room

Solution Module

Output: queries to run on local RSP engine
**EXAMPLE – OVERVIEW OF INPUTS**

**Reasoner goal**: look for a Fault

```prolog
{?x a :Fault.} => {?x a :Fault.}.
```

**Ontology – relevant definitions (Manchester syntax) & individuals:**

```prolog
:LightIntensityAboveThresholdFault ≡ (,:hasSymptom some :LightIntensityAboveThresholdSymptom) and ...
:LightIntensityAboveThresholdFault ⊆ :Fault

:Concussion a :Diagnosis; :hasMedicalSymptom :ConcussionSensitivenessToLight.
:ConcussionSensitivenessToLight :hasThreshold :ConcussionLightThreshold.
:ConcussionLightThreshold :hasDataValue "170"^^xsd:float;
  :isThresholdOnProperty [ a :LightIntensity ].
```

**Context – part about Bob and his room:**

```prolog
:Bob a :Person; :hasRole [ a :PatientRole ]; :hasLocation :R101; :hasDiagnosis :Concussion.
:40-a5-ef-05-a4-a6-A0 a :SoundSensor; :hasLocation :R101.
:40-a5-ef-05-a4-a6-A1 a :LightSensor; :hasLocation :R101.
:40-a5-ef-05-a4-a6-A2 a :TemperatureSensor; :hasLocation :R101.
...
WHAT IS THIS SENSOR QUERY RULE?

{  
  ?p :hasRole [ a :PatientRole ] ;  
  :hasLocation ?l ;  
  :hasDiagnosis [ :hasMedicalSymptom [ :hasThreshold [  
    :hasDataValue ?t ; :isThresholdOnProperty [ a ?prop ]  

  ?s a :Sensor ; :observes [ a ?prop ] ; :hasLocation ?l .
} => {  
  _:x a :Query ;
  :queryPattern { _:o a sosa:Observation ; :madeBySensor ?s ;  
    :hasResult [ :hasDataValue _:v ] . } ;
  :valueThreshold ?t .

  _:oo a :observation ; :madeBySensor ?s ;
  :hasResult [ :hasDataValue _:v ] ;
} .

Representation of generic query
MAGIC BEHIND SOLUTION MODULE

Proof-of-Concept: N3Logic & EYE reasoner

Reasoner inputs:
- Ontology + sensor query rule + context
- Rules (OWL-RL & existential rules supported by N3 not in OWL-RL)

Reasoner goal: look for a Fault

\{?x a :Fault.\} => {?x a :Fault.}.

⇒ EYE reasoner produces a proof with the goal as the last applied rule
WHAT IS THIS SENSOR QUERY RULE?

{  
  ?p :hasRole [ a :PatientRole ] ;  
  :hasLocation ?l ;  
  :hasDiagnosis [ :hasMedicalSymptom [ :hasThreshold [  
    :hasDataValue ?t ; :isThresholdOnProperty [ a ?prop ]  

  ?s a :Sensor ; :observes [ a ?prop ] ; :hasLocation ?l .
}

=> {  
  _:x a :Query ;  
  :queryPattern {  
    _:o a sosa:Observation ; :madeBySensor ?s ;  
    :hasResult [ :hasDataValue _:v ] ; } ;  
  :valueThreshold ?t .

  _:oo a :observation ; :madeBySensor ?s ;  
  :hasResult [ :hasDataValue _:v ] ;  
} .

Representation of generic query
HOW ARE THE QUERIES DERIVED?

Instantiated relevant queries appear in this proof

Can be extracted with additional reasoning step

```triple
<#lemma32> a r:Inference;
  r:gives {
    _:sk_11 a ns4:Query.
    _:sk_11 ns4:queryPattern {
      _:sk_12 a sosa:Observation.
      _:sk_12 sosa:madeBySensor
      _:sk_12 sosa:hasResult __:sk_13.
      _:sk_13 a SSNiot:QuantityObservationValue.
      _:sk_13 DUL:hasDataValue __:sk_14}.
    _:sk_11 ns4:valueThreshold "170.0"^^xsd:float.
    _:sk_15 a sosa:Observation.
    _:sk_15 sosa:madeBySensor
    _:sk_15 sosa:hasResult __:sk_16.
    _:sk_16 a SSNiot:QuantityObservationValue.
    _:sk_16 DUL:hasDataValue __:sk_17.
    _:sk_15 SSNiot:hasSymptom __:sk_18.
    _:sk_18 a SSNiot:ThresholdSymptom.
    _:sk_18 ssn:forProperty __:sk_19.
    __:sk_19 a SSNiot:LightIntensity.
  }
  r:evidence (  
    <#lemma46>  
    ...
    <#lemma57>  
  );
  r:rule <#lemma58>.
```
What is the real-time data flow?

**Classical approach**
- All sensor data
- Semantic reasoner (e.g. RDFox)
- Detected faults after (incremental) reasoning

**Our new approach**
- All sensor data
- RSP engine (e.g. C-SPARQL)
- Detected faults by simple pattern matching (no real-time reasoning required anymore)
FUTURE STEPS IN THIS RESEARCH

1. Building a first Proof-of-Concept on an extended version of the presented use case
2. Introducing SHACL for the query representation
3. Solving the challenges related to retrieving an RSP engine ready query from the proof
4. Evaluation: comparison with real-time reasoning approach (RDFox?)
ir. Mathias De Brouwer
PhD Student

Ghent University – imec
IDLab – Internet Technology and Data Science Lab

E  mrdbrouw.DeBrouwer@UGent.be
T  +32 9 331 49 38
M  +32 484 97 43 15

http://idlab.ugent.be
http://idlab.technology