# *StreamJess*: A Stream Reasoning Framework for Water Quality Monitoring

Edmond Jajaga

Department of Computer Science,
South East European University,
Tetovë, Macedonia
Email: e.jajaga@seeu.edu.mk

Lule Ahmedi

Department of Computer Engineering,
University of Prishtina,
Prishtinë, Kosova
Email: lule.ahmedi@uni-pr.edu

Figene Ahmedi*

Department of Hydro-Technic,
University of Prishtina,
Prishtinë, Kosova
Email: figene.ahmedi@uni-pr.edu
*Corresponding author

**Abstract:** Stream data knowledge bases modeled with OWL are a proved natural approach. But, querying and reasoning over these knowledge bases is not supported with standard Semantic Web technologies like SPARQL and SWRL. Query processing systems enable querying, but to the best of our knowledge, Semantic Web rules are still unable to handle the required reasoning features for effective inference over stream data i.e. non-monotonic, closed-world and time-aware reasoning. In absence of such system, we showed in our previous work how Jess can be used for monitoring water quality, but by bringing input data manually. In this paper, we enable stream data support and thus a timely detection of faulty water quality statuses. The system also identifies the potential sources of pollution by also extending our ontology with the pollutants module. The solution utilizes C-SPARQL abilities to filter and aggregate RDF streams on windows to enable closed-world and time-aware reasoning with Jess rules. Moreover, Jess Tab functions are used to enable non-monotonic behavior.

**Keywords:** stream data; expert system; reasoning; Jess; rules;

**Reference** to this paper should be made as follows: Jajaga, E., Ahmedi, L. and Ahmedi, F. (2016) 'StreamJess: A Stream Reasoning Framework for Water Quality Monitoring', *Int. J. Metadata, Semantics and Ontologies*, Vol. X, No. Y, pp.000–000.

## 1 Introduction

Sensor measurements, social networks, health monitoring, smart cities and other massive data sources have influenced a technological shift to a new concept known as Big Data (Computing Community Consortium, 2011). As a result, Big Data applications should be able to quickly consume volumes of these data and immediately infer as much knowledge as possible. The only paradigm which offers widely-accepted standards and tools for meeting these requirements is the Semantic Web. Recently, a new research area, Stream Reasoning (SR), has evolved. It utilizes Semantic Web techniques for reasoning with stream data (Della Valle et al., 2008). Stream data are defined as unbounded sequences of time-varying data elements (Della Valle et al., 2008).

Semantic technologies have been successfully applied to stream data domains (Jajaga et al., 2013). OWL ontologies have been widely used for modeling stream data domains, e.g., the SSN ontology (Compton et al., 2012). Querying these knowledge bases has been merely done by SPARQL extensions e.g. C-SPARQL (Barbieri et al., 2010), EP-SPARQL (Anicic et al., 2011), etc. However, the windows opened over streams can determine changes in the static information sources, which cannot be applied by query systems (Tallevi-Diotallevi et al., 2013). For example, if a sensor reading the values of a particular water quality parameter provides critical values, then the measurement site's status should be modified as "polluted". Rules provide an effective mechanism for applying changes on the knowledge base induced by streams. Although layering different rule systems over ontologies has already been suggested (Jajaga and Ahmedi, 2015), using Semantic Web standard rule languages, SWRL (Horrocks et al., 2015) and RIF (Boley et al., 2007), over stream data has to the best of our knowledge not been considered to date. The ideal SR system would be to implement rule-based reasoning tasks within the Semantic Web platform. In line with this vision, we have previously developed the INWATERSENSE (INWS) ontology (Ahmedi et al., 2013) and an expert system (Jajaga et al., 2015) demonstrating its usage. In this paper we apply Java Expert System Shell (Jess) (Hill, 2003) to reason over stream data sets. We plan to compare this system with the envisioned pure Semantic Web one.

The Jess system is validated with simulated data in the water quality monitoring (WQM) domain, but it is developed for use within the InWaterSense project[1] with real data. InWaterSense is an EU funded research project aimed to apply recent advanced practices stemming from ICT in water quality monitoring for healthy environment, and strengthen Kosovo's capacity in research in national priority sectors of environment and ICT. An intelligent wireless sensor network (WSN) for monitoring surface water quality has been deployed in a river in Kosova (Ahmedi et al., 2013; 2015; Jajaga et al., 2013; 2015), and is further being enriched with more intelligent behavior like is the contribution presented in this paper.

The paper is organized as follows. Section 2 describes the motivation of building our Jess expert system *StreamJess*. The main contribution of the paper is presented in Section 3 and 4 which exhibits the system design and its implementation, respectively. System validation is presented in Section 5 through examples in the domain of WQM. Section 6 presents *StreamJess* system challenges and related works together with the discussion of building a pure Semantic Web as a future prospect. Finally, the paper closes with conclusion and future plans.

## 2 Problem statement

Stream data domains differ from other Semantic Web ones because of the high frequency of changes in the knowledge base. As a consequence, a SR system should not only reason over background data but also over real-time streaming data. This distinction makes the reasoning tasks hard to implement in Semantic Web. Namely, the Semantic Web rules run over all information present in the knowledge base. In SR this inadequacy has been solved with the introduction of the notion of windows. A window extracts the last data stream elements, be it physical (a given number of triples) or logical (a number of triples occurring in a given time interval) (Barbieri et al., 2010).

SR query processing systems are effectively giving a real-time perception of the situation. As query systems they do not support or are limited on performing ontology modifications. As a consequence, they do not allow using previous queried results for further reasoning. Thus, it is hard to analyze historical events. On the other side, the firing of rules will continually publish new information in ontology, from which can be further inferred new knowledge. This way the resulting inferences can be eventually archived in the form of historical data.

Regarding the reasoning features, a SR rule-based system should support closed-world, non-monotonic, incremental and time-aware reasoning.

### 2.1. Monotonicity

Semantic Web technologies provide a good basis for modeling different domains of discourse. Since the Web is open and accessible by everyone, Semantic Web languages (OWL and SWRL) manage knowledge in terms of open world assumption (OWA). In OWA, if some knowledge is missing it is classified as undefined, as opposed to the closed-world assumption (CWA) which classifies the missing information as false. In the Web, addition of new information does not change any previously asserted information which is known as monotonic reasoning. This is not the case with non-monotonic reasoning in which addition of new information implies eventual modifications in the knowledge base. OWL and SWRL's OWA and monotonic reasoning in Stream Reasoning application domains do not offer the desired expressivity level. For example, modifying the river pollution status is not allowed through SWRL rules. Following the SWRL's monotonic nature a river instance firstly asserted as "clean" cannot be later modified to "polluted".

Non-monotonic operators, aggregates and negation, are common requirements for processing data streams (Zaniolo, 2012). Aggregate operations are present in almost every rule for classifying water bodies into corresponding statuses (Statutory Instruments, 2011) e.g. finding arsenic observations' average value. OWA's approach means one cannot "close" the world to calculate an average value and thus CWA will be a preferable approach. One can use SQWRL (O'Connor and Das, 2009), a SWRL-based OWL query language, constructs such as `sqwrl:average`, but, that approach is not supported. Using SQWRL constructs in SWRL rules for asserting new knowledge is not allowed (an answer received on the Protégé mailing list).

Additionally, a number of example rules need to infer new knowledge in absence of a fact, the concept known as negation as failure (NAF), which is based on the closed-world assumption. For example, the rule "assign 'undetermined status' to those remaining bodies of water where the agency is not, by that date, in a position to assign a reliable interim

[1] http://inwatersense.uni-pr.edu/

classification due to a lack of data or other reason" (Statutory Instruments, 2011) cannot be expressed in SWRL.

### 2.2. Incremental reasoning

Pre-computing and storing of implicit ontology entailments is a process known as materialization. Every time a change occurs, a new materialization need to be computed, which in Semantic Web is known as incremental maintenance of materialization (Volz et al., 2005). As opposed to traditional Semantic Web applications, in SR ones change to the facts occurs "regularly" i.e. new facts are added and other ones updated or deleted. A technique for computing ontological entailments on stream reasoning is presented in (Barbieri et al. 2010). It uses Logic Programming, respectively Datalog rules, to compute incremental materialization for window-based changes of ontological entailments. This approach is concerned with computing complete and correct materialization enforced by changes to facts.

According to (Volz et al., 2005), there is another type of incremental materialization which needs to be addressed in rule-enabled ontologies. Namely, changes to the ontology will typically require changes in the rules. Authors of (Volz et al., 2005) describe a technique of this type of incremental materialization. The frequency of changes to the ontology in SR applications is the same as in traditional Semantic Web ones. Therefore, the techniques developed for this type of incremental materialization intended for "static" knowledge bases would also be suitable for stream data knowledge bases.

### 2.3. Time-aware reasoning

SR systems should include time-annotated data i.e. the time model, and like Complex Event Processing (CEP) (Luckham and Schulte, 2011) should offer explicit operators for capturing temporal patterns over streaming information (Margara et al., 2014). INWS ontology implements the time model through OWL Time ontology. Supporting temporal operators (serial, sequence etc.) means the system can express the following example rule: Enhanced phosphorus levels in surface waters (that contain adequate nitrogen) can stimulate excessive algal growth (Foundation for Water Research, 2005). If before excessive algal growth, enhanced phosphorus level has been observed then more probably the change of phosphorus levels has caused the algal growth. Thus, a sequence of these events needs to be tracked to detect the occurrence of this complex event.

Moreover, in order to enable reasoning in terms of time and quantity intervals of continuous and possibly infinite streams the SR notion of windows needs to be adapted for rules (Mileo et al., 2013). In traditional settings, rules operate over all asserted facts in the ontology. This is not practical with stream data as data flow is massive and rules may not always consider all RDF streams. Instead, rules must be evaluated against a certain set of RDF streams which will also reduce information load. For example, a rule to assert which sensors provided observation measurements that are above allowed average threshold the last 3 minutes sliding the window every minute, will be easily expressible with the help of the window. With each window processing new logical decisions will arise: new information need to be published on the knowledge base or a fact modification/retraction need to be performed.

## 3 Conceptual architecture

The conceptual architecture of *StreamJess* is depicted in Figure 1. It consists of four layers: data, ontology, stream filtering and aggregation, and rules layer. Domain-specific data (blue track left) and data streams (blue track right) constitute the data layer. The green track of the figure represents the ontology model. A module for filtering and aggregating stream data is represented by the brown track. Rules (pink track) in *StreamJess* mainly fall into two broad categories:

- *monitoring* rules (pink track left), rules for continuous classification of water bodies based on in situ observations, and
- *investigation* rules (pink track right), which fire after monitoring rules detect any critical status. The information of sources of pollution stored into the pollutants ontology is used to prejudge the causer of the pollution.

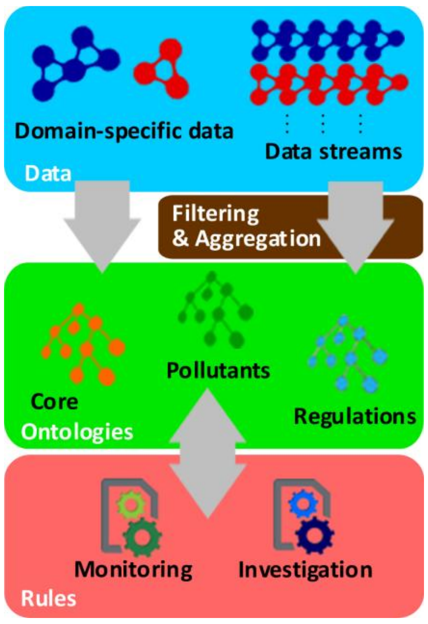


Figure 1. *StreamJess* conceptual architecture

Both kinds of rules are loaded at system start up. Grey arrows describe data flow direction. Our system acts as a pipeline. Sensor produced or simulated data streams are firstly filtered and aggregated within a time or tuple-based window and then the output results are published as observation data in the working memory and on the ontology. The running rule engine indicates the facts change and infers new knowledge according to the preloaded rules. In our case of study, incoming water quality measured values are filtered and aggregated within a window. The calculated results are used by monitoring rules to continually classify the observations within the appropriate water statuses. Whenever a critical status becomes detected, appropriate investigation rule acts to identify the potential source(s) of pollution.

## 4 Implementation

*StreamJess* is implemented as a Java console application. The application uses an instance of `jess.Rete` which is created at system start up. It provides the central access point of the application as it loads the ontology, builds the working memory, holds the list of rules and offers the methods for doing CRUD operations over facts i.e. ontology individuals (Hill, 2003). The module for stream data filtering and aggregation is implemented with a well-known stream processing system C-SPARQL (Barbieri et al., 2010). Multiple C-SPARQL queries and Jess rules can be defined to process RDF streams and reason over them.

*StreamJess* is open for loading other SR domain ontologies and write appropriate C-SPARQL queries and Jess rules.

### 4.1. Ontology layer

Ontologies are defined as formal specification of a shared conceptualization (Gruber, 1993). They have been extensively used for modeling stream data domains. In our previous work (Ahmedi et al. 2013), we have built the INWS ontology, an ontology framework for modeling WQM systems. INWS ontology consists of three ontology modules: core, regulations and pollutants. The core ontology is a SSN-based ontology (Compton et al., 2012) which models WSN infrastructure entities, observations and water quality parameters. The regulations ontology models classification of water bodies based on different regulation authorities such as Water Framework Directive (WFD) (European Parliament and Council of Europe, 2000). Finally, the pollutants ontology models the entities for investigating sources of pollution.

Protégé functions of Jess in Jess Tab (Jess Wiki) were used to manage with the knowledge base. Jess Tab is a plug-in for the Protégé ontology editor and knowledge-engineering framework that allows one to use the Java Expert System Shell (Jess) and Protégé together (Jess Wiki). All ontology modules are imported and loaded at application start up. Moreover, ontology class instances are also mapped into the Jess's working memory and `java.util.Random` class is imported for generating random numbers, which are used for creating unique instance names.

#### 4.1.1. The Pollutants Ontology

The INWS pollutants ontology was designed based on examples of sources of pollution and the potential pollutant discharges which could arise described in (Foundation for Water Research, 2005). As depicted in Figure 2, two classes were created: `PollutionSources`, describing the sources of pollution e.g. urban storm water discharges, and Pollutants, representing contaminants present in the environment or which might enter the environment which, due to its properties or amount or concentration, causes harm e.g. heavy metals. A property `potentialPollutant` links individuals of `PollutionSources` and `Pollutants` [Foundation for Water Research, 2005, p.3]. `PollutionSources` class is also linked with a string through two properties: `pollutionSourceName`, representing the name of the pollution source, and `pollutionType`, representing the type of the pollution source which can be point, diffuse or both of them. Moreover, a property `hasSourcesOfPollution` was added to relate river's measurement sites with the sources of pollution.

### 4.2. Data layer

In general, there are two kinds of data that SR applications deal with: domain specific ABox data which do not change or change "slowly" that are formulated in the form of RDF data e.g. river names, and stream data e.g. sensor observed data. In *StreamJess* RDF data are specified by end-users and populate the corresponding ontology modules before system start up.

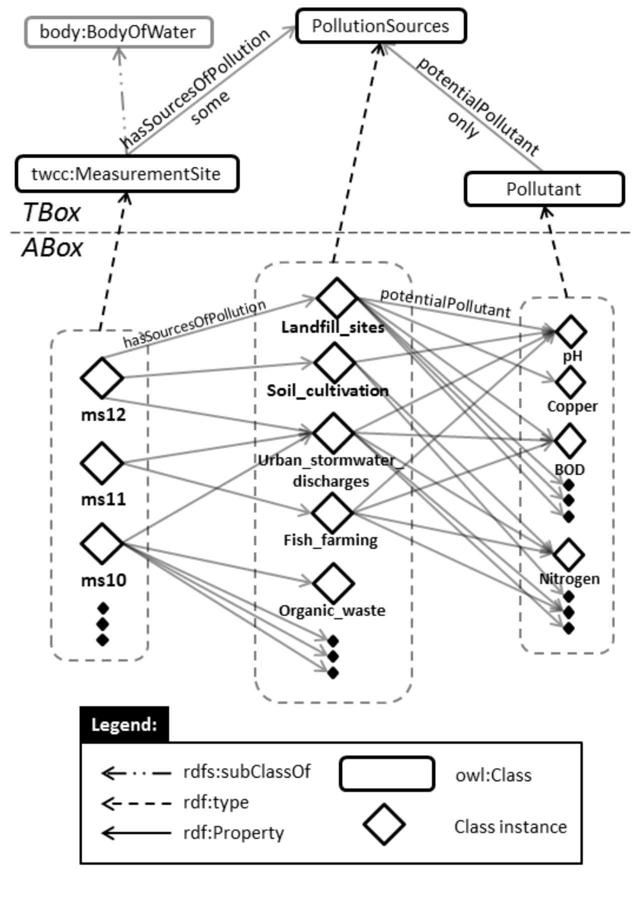


Figure 2. TBox and ABox statements for the INWS pollutants ontology module

Query processing systems (e.g., C-SPARQL and EP-SPARQL) model stream data in the form of RDF streams. RDF streams are defined as a sequence of RDF triples that are continuously produced and annotated with a timestamp (Barbieri et al., 2010). For example in *StreamJess*, a single RDF stream holds information of a measured water quality name and value coupled with timestamp and location. In a real scenario the input RDF streams are generated by sensor probes deployed in different measurement sites. Here, as per validation purposes, we use simulated and randomly generated values.

### 4.3. Stream processing

A C-SPARQL engine is also initialized at application start – up to enable filtering and aggregation of data streams in logical or physical windows. We have currently implemented two queries, one for considering RDF streams one by one and another one using aggregation functions. However, other queries can be easily encoded and run simultaneously. After running the application, the user is queried to specify the type of the windows and their parameter values. If one prefers physical windows then he/she will have to specify the size of the window by entering the number of tuples. Otherwise, one can choose logical windows and thus specifying the window size by giving the number of seconds. In the case of time-based windows, the default value for intervals between windows is set `TUMBLING` i.e. non-overlapping windows.

Each C-SPARQL query in *StreamJess* eventually outputs triples of values: the water quality name, the location of measurements and the calculated value. Every output triple is stored into the knowledge base and mapped into a temporary observation class. Thus, C-SPARQL enables time-aware and closed-world reasoning. The non-monotonic reasoning is complemented by the reasoning engine, described in the next subsection. The processing engine runs independently from the reasoning engine.

### 4.4. Stream reasoning

Before implementing *StreamJess*, in order to enable Jess rules to reason over stream data, three approaches were considered:

- Extending Jess with stream data reasoning features,
- Translating Jess to another rule system which supports stream data reasoning and
- Layering Jess on top of another system to fill the gaps of Jess in support of stream data reasoning.

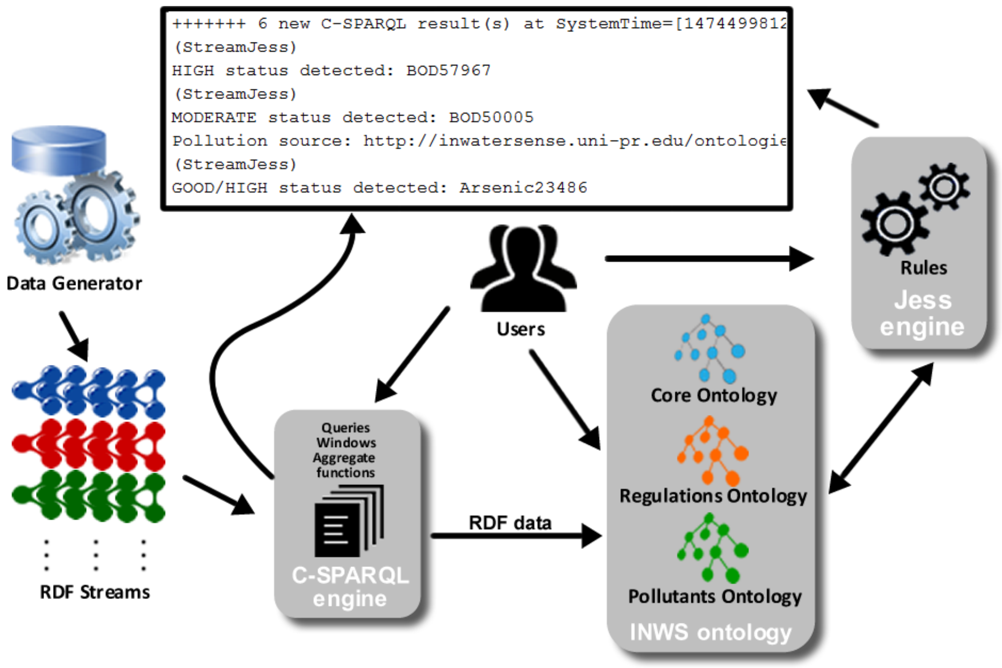
Extending Jess with stream data reasoning features is very expensive. Event stream processing with Jess is a fragile system, the code is complex and a lot of interferences have to be taken into account (Ermert, 2009). As the author of (Ermert, 2009) argues, code could not be optimized even for simple temporal operations over event-streams. Another approach would be to translate Jess constructs into any CEP system. To the best of our knowledge there is not any evidence of such an approach. Albeit of the translation overhead we do not have confidence of how the system would perform.

Given the drawbacks if approaching any of the previous two options, it was decided to layer Jess over an existing stream processing system such as C-SPARQL. However, as a query language, it is not intended to have any effect on the underlying ontology. In *StreamJess* we use Jess rules for populating the knowledge base. Moreover, they enable data modifications i.e. non-monotonic reasoning and the tools for archiving data.

After C-SPARQL processes the window and publishes new observation results, a new call to the Rete method `run()` is invoked for doing rule-based reasoning. As illustrated in Figure 3, the Jess engine runs the rules against the newly published temporary observation facts and it eventually activates the rule's RHS actions. The inferred knowledge forms another set of RDF data which is stored back into the ontology for further reasoning. Namely, monitoring rules do the water quality classifications based on the WFD regulation rules. In case a critical status is detected i.e. new instance of a 'moderate' status class gets published, investigation rules act to identify the pollution source. Namely, it prints out the names of the pollution sources present on the observation's measurement site stored on the background knowledge. Furthermore, it archives the instance of the status for historical purposes.

## 5 Examples of *StreamJess*

As a proof of concept, we have implemented *StreamJess* in a typical WQM scenario based on WSN. Sensors in InWaterSense WQM system are deployed in different measurement sites at different times. They continually emit water quality values. *StreamJess* will (1) classify the water body into the appropriate status according to WFD regulations (Environment Agency, 2011; Statutory Instruments, 2009) and (2) identify the potential sources of pollution if the pollutants values are out of the allowed thresholds. In general, each water quality is monitored and investigated with a monitoring rule and an investigation one. A couple of examples are used to validate the system performance. Both examples run at the same time over the same RDF streams which are filtered out by two different C-SPARQL queries: one for finding the average values of water quality observations and another one for considering observation values one by one. The simulator was set up to randomly generate observation data for an arbitrary number of 70 measurement sites and 11 water quality parameters. A single sensor observation was arbitrarily set to be produced every second and includes 6 RDF streams representing time, location, device and quality of observation information. For example, in a 20 seconds window size 120 tuples will be produced.



```
+++++++ 6 new C-SPARQL result(s) at SystemTime=[1474499812
(StreamJess)
HIGH status detected: BOD57967
(StreamJess)
MODERATE status detected: BOD50005
Pollution source: http://inwatersense.uni-pr.edu/ontologie
(StreamJess)
GOOD/HIGH status detected: Arsenic23486
```

Figure 3. *StreamJess* system workflow

Moreover, the system supports registering multiple streamers to run concurrently.

5.1. Example 1: pH observations

A WFD rule for classifying pH observations looks as follows: The pH as individual value should be between 4.5 and 9.0 (Statutory Instruments, 2011). Potential sources of pollution from which pH discharges could arise include: agricultural fertilizers, farm wastes and silage, effluent discharges from sewage treatment works, fish farming, organic waste recycling to land, soil cultivation and urban storm water discharges (Foundation for Water Research, 2005).

A simple C-SPARQL query to filter out incoming pH observations, i.e. pH RDF streams, is described below:

```
1   REGISTER QUERY IndObservations AS
2   PREFIX inws: <http://inwatersense.uni-
    pr.edu/ontologies/inws-core.owl#>
3   PREFIX ssn:
    <http://purl.oclc.org/NET/ssnx/ssn#>
4   PREFIX dul: <http://www.loa-
    cnr.it/ontologies/DUL.owl#>
5   SELECT ?qo ?loc ?dv
6   FROM STREAM <http://inwatersense.uni-
    pr.edu/stream> [RANGE 10s STEP 10s]
7   WHERE {
8   ?o ssn:qualityOfObservation ?qo .
9   ?o ssn:observationResult ?r .
10  ?r ssn:hasValue ?v .
11  ?v dul:hasDataValue ?dv .
12  ?o inws:observationResultLocation ?loc
13  FILTER (?qo = inws:pH)
14  }
```

The query name is registered on line 1 and prefixes used in the query are declared on lines 2, 3 and 4. The query runs against the input RDF streams in the time frame of 10 seconds, sliding the window by 10 seconds (line 6). The chosen time frame is arbitrary and can be changed as desired. It produces triples of values (line 5): the water quality name (`?qo`), the location of measurements (`?loc`) and the observation value (`?dv`). Based on the INWS metadata descriptions the incoming observation's (`?o`) water quality name is saved on variable `?qo` (line 8). To get the observation's value, `?o` individuals are bound with individuals `?r` through `ssn:observationResult` property (line 9). These ones in turn are related with individuals of class `ssn:ObservationValue` (line 10), which are finally related with the data value `?dv` through `dul:hasDataValue` property (line 11). The location of observations `?loc` is get through `inws:observationResultLocation` property. Finally, the list of observations is filtered out to include only pH observations (line 13).

Output query results, i.e. (`?qo, ?loc, ?dv`) triples, are consumed by Jess Tab functions for asserting new facts into the knowledge base. `make-instance` and `slot-insert$` functions are used for creating new class individuals and inserting property values respectively. Namely, for every outputted triple, a new observation instance of the temporary class `tmpObservation` (a subclass of the `ssn:Observation` class) is created. `tmpObservation` holds the most current observation data which are retracted after *StreamJess* rules process them. Moreover, after retraction they are archived in the `ssn:Observation` class in the form of historical data. The newly created observation individual is further related with `?qo,` `?loc` and `?dv` values based on the structure of the SSN and INWS metadata descriptions. The water quality name `?qo` i.e. pH, becomes related with the new observation instance through the `ssn:qualityOfObservation` data property. The new observation instance also becomes related with `?loc` through `observationResultLocation` object property. The location instance is of type `Point` of the basic geo location vocabulary, which means that it possesses longitude and latitude properties. A new `ssn:SensorOutput` individual is also created for holding the observed value `?dv`. It is linked with the observation instance through `ssn:observationResult` property. Meanwhile, a new instance of class `ssn:ObservationValue` is created to be related with the previously created `ssn:SensorOutput` individual through `ssn:hasValue` property. The `?dv` value is assigned to it through `dul:hasDataValue` data property.

To implement the scenario of this example a monitoring rule was designated for deciding the pH status and another one for identifying the eventual sources of pollution. The monitoring rule looks like follows (ontologies' full IRI are omitted for brevity):

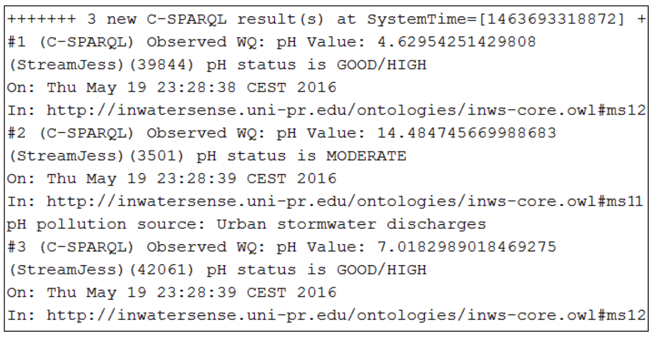
```
1 (defrule classifyPHObsValues
2 (declare (salience 54))
3 (object (is-a ssn#ObservationValue)
4 (OBJECT ?ov)(DUL.owl#hasDataValue ?x))
5 (object (is-a ssn#SensorOutput)
6 (OBJECT ?so)(ssn#hasValue ?ov))
7 (object (is-a time#Instant)
8 (OBJECT ?ot)(time#inXSDDateTime ?time))
9 (object (is-a inws-core.owl#tmpObservation)
10(OBJECT ?o)(ssn#observationResult ?so)
11(inws-core.owl#observationResultLocation
  ?loc)(ssn#observationResultTime ?ot)
12(ssn#qualityOfObservation ?qo&:(eq (in-
  stance-name ?qo) inws-core.owl#pH)))
13=>
14(bind ?*r* (random))
15(printout t "(StreamJess)")
16(if (and (> ?x 4.5) (< ?x 9))then (and
17(printout t "(" ?*r* ") pH status is
  GOOD/HIGH" crlf "On: " ?time crlf "In: "
  (instance-name ?loc) crlf)
18(make-instance (str-cat "GoodHighPHStatus"
  ?*r*) of inws-
  regulations.owl#GoodHighPHMeasurement map)
19(slot-insert$ (str-cat "GoodHighPHStatus"
  ?*r*) inws-
  core.owl#observationResultLocation 1 ?loc)
20(slot-insert$ (str-cat "GoodHighPHStatus"
  ?*r*) ssn#observationResultTime 1 ot)
21(slot-set ?loc
  inws-regulations.owl#isPolluted FALSE))
22else (and
23(printout t "(" ?*r* ") pH status is
  MODERATE" crlf "On: " ?time crlf "In: "
  (instance-name ?loc) crlf)
24(make-instance (str-cat "ModeratePHStatus"
  ?*r*) of inws-
  regulations.owl#tmpModeratePHMeasurement
  map)
25(slot-insert$ (str-cat "ModeratePHStatus"
  ?*r*) inws-
  core.owl#observationResultLocation 1 ?loc)
26(slot-insert$ (str-cat "ModeratePHStatus"
  ?*r) ssn#observationResultTime 1 ?ot)
27(slot-set ?loc
  inws-regulations.owl#isPolluted TRUE)))
28(make-instance (str-cat (instance-name ?o)
  ?*r*) of ssn#Observation map)
29(slot-insert$ (str-cat (instance-name ?o)
  ?*r*) inws-
  core.owl#observationResultLocation 1 ?loc)
30(slot-insert$ (str-cat (instance-name ?o)
  ?*r*) ssn#observationResult 1 ?so)
31(slot-insert$ (str-cat (instance-name ?o)
  ?*r*) ssn#observationResultTime 1 ?ot)
32(slot-insert$ (str-cat (instance-name ?o)
  ?*r*) ssn#qualityOfObservation 1 inws-
  core.owl#pH)
(unmake-instance ?o))
```

The first line serves for declaring rule's definition and asserting its name. The second one is for declaring the rule priority. The left hand side (LHS) of the rule (lines 3-12) matches all pH observation individuals (`?o`) present in the `tmpObservation` class. The right hand side (RHS) of the rule (lines 14-31) asks if the matched observation value (`?x`) falls between the interval of values 4.5 and 9. If so, the observation is classified in "good/high" status (lines 16-21), otherwise it becomes "moderate" (lines 22-27). After the classification takes place the observation individual is stored in the `ssn:Observation` class (lines 28-32) and the temporary observation individual (`?o`) gets retracted from the knowledge base (line 33).

Concretely, for each matched individual from temporary observation class `?o`, on the RHS a new random value is generated to be used for new individual names (line 14). An information string is printed out in the console to indicate that the upcoming outputs are processed by *StreamJess* rules (line 15). The code in line 16 asks whether the observation value `?x` falls between the allowed values for "good/high" status. If so, the user gets informed about the status detected at measurement site `?loc` on time `?time`. Next, a new individual of `GoodHighPHMeasurement` class gets created (line 18) and related with the location (line 19) and time (line 20) of measurement. Moreover, the pollution status of the measurement site `?loc` is modified to "clean" by changing its `isPolluted` value to "false" (line 21). `str-cat` command is used to concatenate strings. If the if condition specified on line 16 fails then the actions for specifying "moderate" status are activated. The steps to do this are analogical to the ones used for specifying "good/high" status. Namely, before setting the status of the measurement site as "polluted" (line 27) the new status instance is created to be of type `tmpModeratePHMeasurement` (line 24). These instances are temporary because the investigation rule to find potential pH sources of pollution will make use of them and after that will delete them. Prior to deletion the status instance is stored as a new instance of `ModeratePHMeasurement` class as historical data (lines 28-32) copying all `?o` properties. The retraction is performed for preventing investigations to be activated only once. The pH investigation rule is described below:

```
1  (defrule findPHsourcesOfPollution
2  (declare (salience 553))
3  (object   (is-a   epa.owl#MeasurementSite)
   (OBJECT        ?loc)         (inws-
   pollutants.owl#hasSourcesOfPollution
   $?sitepoll))
4  (object          (is-a          inws-
   regulations.owl#tmpModeratePHMeasurement)
   (OBJECT        ?mob)         (inws-
   core.owl#observationResultLocation   ?loc)
   (ssn#observationResultTime ?ot))
5  =>
6  (bind ?*r* (random))
7  (make-instance   (str-cat   (instance-name
   ?mob)       ?*r*)        of        inws-
   regulations.owl#ModeratePHMeasurement map)
8  (slot-insert$ (str-cat (instance-name ?mob)
   ?*r*)                              inws-
   core.owl#observationResultLocation 1 ?loc)
9  (slot-insert$ (str-cat (instance-name ?mob)
   ?*r*) ssn#observationResultTime 1 ?ot)
10 (foreach ?poll ?sitepoll
11  (foreach ?pollLsItem (slot-get ?poll inws-
   pollutants.owl#potentialPollutant)
12   (if(eq (instance-name ?pollLsItem) inws-
   core.owl#pH) then
13    (printout  t  "pH  pollution  source:  "
   (instance-name ?poll) " crlf)
14    (slot-insert$  (str-cat  (instance-name
   ?mob)          ?*r*)             inws-
   regulations.owl#foundPollutionSources    1
   (instance-name ?poll)))))
15 (unmake-instance ?mob))
```

The rule binds the temporary “moderate” status pH observations into `?mob` variable and gets its location `?loc` and time `?ot` (line 4). The code in line 3 relates the list of sources of pollution present on the measurement site `?loc` into the list variable `$?sitepoll`. The RHS of the rule starts with archiving the temporary status instance `?mob` (lines 6-9). Namely, in absence of a Jess or Jess Tab mechanism to change the instance class assignment, the temporary status instance is copied in a new instance of class `ModeratePHMeasurement`. Afterwards, the list members of `?sitepoll` is iterated (line 10) to match only those sources of pollution which could increase pH discharges (lines 11-12). Namely, for each source of pollution in `?sitepoll` i.e. present on the measurement site, its potential pollutants list `?pollLsItem` is checked if it includes pH. The matching one’s name will be printed out (line 13). As per saving historical data the archived status instance gets related with the list of pollution sources through `foundPollutionSources` property (line 14). Finally, the temporary status instance `?mob` gets discarded from the knowledge base. An example output of *Example 1* is illustrated in Figure 4. As can be observed, C-SPARQL query `IndObservations` has produced three output results. Two of these results (#1 and #3) have been classified with “good/high” status by rule `classifyPHObsValues`, while the remaining one (#2) with “moderate” status. Since the result #2 has been classified as a critical status the investigation rule `findPHsourcesOfPollution` has fired and identified that potential source of the pollution is “urban storm water discharges” on site `ms11`.



```
+++++++ 3 new C-SPARQL result(s) at SystemTime=[1463693318872] +
#1 (C-SPARQL) Observed WQ: pH Value: 4.62954251429808
(StreamJess)(39844) pH status is GOOD/HIGH
On: Thu May 19 23:28:38 CEST 2016
In: http://inwatersense.uni-pr.edu/ontologies/inws-core.owl#ms12
#2 (C-SPARQL) Observed WQ: pH Value: 14.484745669988683
(StreamJess)(3501) pH status is MODERATE
On: Thu May 19 23:28:39 CEST 2016
In: http://inwatersense.uni-pr.edu/ontologies/inws-core.owl#ms11
pH pollution source: Urban stormwater discharges
#3 (C-SPARQL) Observed WQ: pH Value: 7.0182989018469275
(StreamJess)(42061) pH status is GOOD/HIGH
On: Thu May 19 23:28:39 CEST 2016
In: http://inwatersense.uni-pr.edu/ontologies/inws-core.owl#ms12
```

Figure 4. An output excerpt of the running *Example 1*

## 5.2. Example 2: Biochemical Oxygen Demand ($BOD_5$) observations

A WFD rule for classifying - $BOD_5$ observations is as follows: If $BOD_5$ measurements in mg $O_2$/l is less than 1.3 (mean), then river belongs to “high” status of oxygen condition; if it is less than 1.5 then river belongs to “good” status of oxygen condition; otherwise the river belongs to “moderate” status of oxygen condition (Statutory Instruments, 2011). Potential sources of pollution from which $BOD_5$ discharges could arise include: contaminated land, farm wastes and silage, fish farming, effluent discharges from sewage treatment works, landfill sites and urban storm water discharges (Foundation for Water Research, 2005).

The C-SPARQL query to calculate the water quality observed average value on each window is given below:

```
1   REGISTER QUERY AvgObservations AS
...
5   SELECT ?qo ?loc (AVG(?dv) AS ?avg)
6   FROM STREAM <http://inwatersense.uni-
    pr.edu/stream> [RANGE 20s STEP 20s]
...
13  FILTER (?qo != inws:pH)
...
15  GROUP BY ?qo ?loc
```

This query is similar to `IndObservations` query described previously. The omitted lines are the same. As opposed to it, this query filters the RDF streams to include all but those of pH observations (line 13). Moreover, it uses aggregate functions such as AVG (line 5) to calculate the average value of observations which are firstly grouped by water quality name and then by location of measurement (line 15). It is arbitrary set to run every 20 seconds sliding the window by 20 seconds (line 6).

Similar to *Example 1*, the output query results, i.e. (`?qo, ?loc, ?avg`) triples, are consumed by Jess Tab functions to create new `tmpObservation` instances. Afterwards, the following monitoring rule, similar to `classifyPHObsValues`, classifies $BOD_5$ observations:

```
1  (defrule classifyBOD5ObsValues
...
12 (ssn#qualityOfObservation ?qo&:(eq (in-
   stance-name ?qo)(instance-name inws-
   core.owl#BOD))))
=>
...
16 (if (and (< ?x 1.5) (> ?x 1.3)) then (and
17 (printout t "(" ?*r* ") BOD status is GOOD"
   crlf "On: " ?time crlf "In: " (instance-
   name ?loc) crlf)
18 (make-instance (str-cat "GoodBODStatus"
   ?*r*) of inws-
   regulations.owl#GoodBODMeasurement map)
19 (slot-insert$ (str-cat "GoodBODStatus"
   ?*r*) inws-
   core.owl#observationResultLocation 1 ?loc)
 ...
20 (slot-insert$ (str-cat "GoodBODStatus"
   ?*r*) ssn#observationResultTime 1 ?ot)
...
22 (if (< ?x 1.3) then <HIGH status classifi-
   cation code here>)
23 (if (> ?x 1.5) then <MODERATE status clas-
   sification code here>))
```

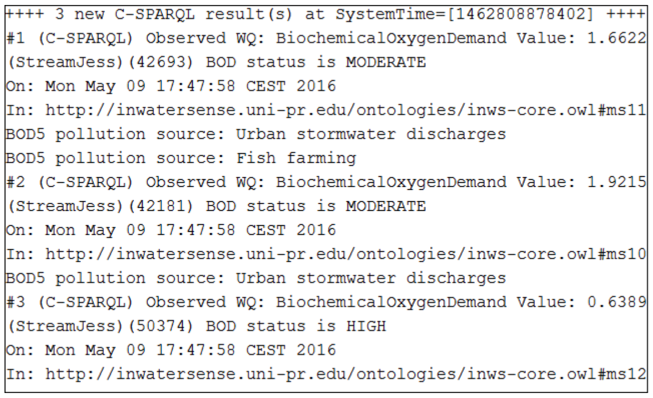
Similarly to the rule `classifyPHObsValues` lines 3-12 bind $BOD_5$ observation data present in the temporary class `tmpObservation` with their corresponding variables. Lines 16-20 encode the semantics of the expression "if it is less than 1.5 then river belongs to "good" status of oxygen condition" from the example statement. Classification of water bodies into "high" (line 22) and "moderate" (line 23) status is omitted because it's analogical with lines 16-20 with the appropriate change on the name of the status, the corresponding class name and the setting of the pollution status of the site.

The streams processed by the C-SPARQL query `AvgObservations` will result in zero or many BOD5 observations. The number of $BOD_5$ observations will depend on the number of measurement sites. For example, as illustrated in Figure 5, C-SPARQL processing of RDF streams has resulted with 3 new observations on 3 measurement sites: `ms10, ms11` and `ms12`. Two observations have been classified as of "moderate" status (Figure 5 line #1 and #2) and one of "high" status (Figure 5 #3).

Whenever a critical i.e. "moderate" $BOD_5$ measurement is detected the following investigation rule to detect $BOD_5$ potential sources of pollution is activated:

```
1  (defrule findBOD5SourcesOfPollution
...
=>
...
7  (make-instance (str-cat (instance-name
   ?mob) ?*r*) of inws-
   regulations.owl#ModerateBODMeasurement map)
...
12 (if(eq (instance-name ?pollLsItem) inws-
   core.owl#BOD) then
13 (printout t "BOD pollution source: " (in-
   stance-name ?poll) crlf)
...
```

Similar to `findPHsourcesOfPollution` rule, this one will cause the Rete engine to detect newly asserted individuals of `tmpModerateBODMeasurement` on a specified measurement site. In fact, the LHS of the rules is the same. It will get the sources of pollution on that site which in turn are filtered out to include only $BOD_5$ potential pollutants (lines 9-13). Each of the matched sources of pollution will be printed out in the console as shown in observation #1 and #2 in Figure 5. Namely, a "moderate" $BOD_5$ status is detected on sites `ms11` and `ms10`. Potential sources of pollution include urban storm water discharges and fish farming on site `ms11` while urban storm water is potential source of $BOD_5$ discharges on site `ms10`.



```
++++ 3 new C-SPARQL result(s) at SystemTime=[1462808878402] ++++
#1 (C-SPARQL) Observed WQ: BiochemicalOxygenDemand Value: 1.6622
(StreamJess)(42693) BOD status is MODERATE
On: Mon May 09 17:47:58 CEST 2016
In: http://inwatersense.uni-pr.edu/ontologies/inws-core.owl#ms11
BOD5 pollution source: Urban stormwater discharges
BOD5 pollution source: Fish farming
#2 (C-SPARQL) Observed WQ: BiochemicalOxygenDemand Value: 1.9215
(StreamJess)(42181) BOD status is MODERATE
On: Mon May 09 17:47:58 CEST 2016
In: http://inwatersense.uni-pr.edu/ontologies/inws-core.owl#ms10
BOD5 pollution source: Urban stormwater discharges
#3 (C-SPARQL) Observed WQ: BiochemicalOxygenDemand Value: 0.6389
(StreamJess)(50374) BOD status is HIGH
On: Mon May 09 17:47:58 CEST 2016
In: http://inwatersense.uni-pr.edu/ontologies/inws-core.owl#ms12
```

Figure 5. An output excerpt of the running *Example 2*

## 5.3. Example 3: The 'undetermined status'

NAF feature is enabled in the stream processing level. Recall sub section 2.1 example of assigning the 'undetermined status' to measurement sites to which the data are missing. The SPARQL support for NAF was utilized as described in the following query.

```
REGISTER STREAM undefinedMeasurmentSites  AS
PREFIX inwsp: <http://inwatersense.uni-
pr.edu/ontologies/inws-pollutants.owl#>
PREFIX rdf: < http://www.w3.org/1999/02/22-
rdf-syntax-ns#>
PREFIX twcc:
<http://tw2.tw.rpi.edu/zhengj3/owl/epa.owl#>
SELECT ?ms
FROM STREAM <http://inwatersense.uni-
pr.edu/stream> [RANGE 60s STEP 60s]
FROM <http://inwatersense.uni-
pr.edu/ontologies/data.rdf>
```

```
WHERE {
?ms rdf:type twcc:MeasurementSite
OPTIONAL { ?ms inwsp:isPolluted ?tf } .
FILTER (!BOUND(?tf))
}
```

After (C-SPARQL) processing and (Jess) reasoning on each observation instance a measurement site will be related with a “true” or “false” value through `isPolluted` property. This query will match the remaining measurement sites, present in the background knowledge base (`data.rdf` file), for which no pollution status is recorded. The query is arbitrarily set to run every minute. On each query output result the matching measurement sites’ `isPolluted` status is set to ‘undefined’ through Jess Tab `make-instance` construct.

## 6 Related Works

Two main strategies exist for systems combining ontologies with rules: hybrid and homogeneous approaches (Jajaga et al., 2013; Eiter et al., 2006). In the former one, also called loosely-coupled approach, the reasoning is done by interfacing existing rule reasoner with existing ontology reasoner, while in the latter one, also called tightly-coupled approach, both ontologies and rules are embedded into the same logical language without making a priori distinction between the rule predicates and the ontology predicates (Eiter et al., 2006).

### 6.1. Hybrid approaches

Hybrid approaches layer different non-DL rule systems on top of DL ontologies like: production rules, CEP, LP, answer set programming (ASP), etc. In the literature this approach is also referred to as, integration of ontologies and rules with strict semantic separation (Eiter et al., 2006). In our previous work (Jajaga et al., 2013), we described in more detail about each one of these approaches and their pros and cons. In general, hybrid solutions have achieved the desired system behavior while main drawbacks include: translation and reasoner issues and side-effects occurrence.

The first approaches combining ontologies with production rules are described in (Sottara et al., 2012; Chau, 2007). Sottara et al. (2012) model a hybrid Environmental Decision Support System (EDSS) for Waste-Water Treatment Plants (WWTP). As an application of production rules they infer invalid $NO_3$ measurement values. They argue that the WWTP domain should be modeled through ontologies, for modeling sensor data, paired with decision-making rules, for processing incoming sensor data and recommending actions to be taken. Another system implemented in terms of production rules has been designed by Chau (2007) in the domain of water quality modeling. Namely, the system simulates human expertise during the problem solving of coastal hydraulic and transport processes. Both forward-chaining and backward-chaining are used collectively during the inference process Chau (2007). Even though that these approaches, together with our previous work (Jajaga et al., 2015) argue that pairing ontologies with production rules provides a fruitful solution, they do not make any distinction between stream and static data. As such, they do not implement the window feature.

StreamRule (Mileo et al., 2013) represents the pioneer of coupling stream processing systems with ASP non-monotonic reasoning. Even though the approach is still much more prototypical it demonstrates how non-monotonic and time-aware reasoning can be integrated into a unique platform for stream data reasoning. Similarly to our approach, the continuous rule feature is implemented through separate steps. Namely, stream filtering and aggregation is done through a stream query processor such as CQELS (Le-Phuoc et al., 2011), while OClingo (Gebser et al., 2013) is used to enable non-monotonic reasoning. In *StreamJess* we use C-SPARQL for filtering and aggregation purposes, while non-monotonic reasoning is achieved through Jess rules and Jess Tab functions. Even though that CQELS outperforms C-SPARQL (Le-Phuoc et al., 2013), we preferred C-SPARQL following its advantage to use nested aggregations and negation (Lanzanasto, 2009; Le-Phuoc et al., 2012). Moreover, we plan to support temporal operators, which lack any support in CQELS (Lanzanasto, 2009). The main distinction of the stream reasoning component between StreamRule and *StreamJess* fall on the strategy of the inference process. Namely, OClingo, as LP-based approach, follows the backward chaining approach. It means to start from the conclusion of the rule and try to match the facts of the rule’s condition part. Jess uses the Rete algorithm (Forgy, 1982) to do fast pattern matching, which is natively forward chaining strategy. Even though the algorithm is ideally suited for complex event detection, it does not support temporal reasoning (Walzer et al., 2008). Moreover, it saves the states between cycles, which is not preferred in situations when most of the data change. However, its extensions are in place to support stream reasoning e.g. (Walzer et al., 2008), (Komazec and Cerri, 2011) and (Schmidt et al., 2008). Jess also supports backward chaining, which is effectively simulated in terms of forward chaining rules (Hill, 2003). The forward chaining technique starts from the rule’s condition part and finds the facts satisfying the rule’s conclusion. Both approaches have their pros and cons: backward chaining is more memory efficient while forward chaining is faster but consumes more memory (Hardy, 2013). We decided to use Jess because of the ability to use both strategies. Regarding the implemented features StreamRule lacks the historical data management component, which is one of the key requirements of SR tools (Margara et al., 2014). *StreamJess* keeps evidence of every previous environment state. For example, one can query the INWS ontology for a particular measurement site’s pollution status of the past. OClingo feeds back the reasoning results into Java runtime for further processing or display, while in *StreamJess*, the results are also deployed back into the knowledge base and thus the memory gets released and the data are available for query and retrieval.

This was implemented through the Jess Tab's save-project function, which is called after processing each C-SPARQL window or alternatively be set to run periodically.

Recently, Ali et al. (2016) describe the descendant of StreamRule, which support C-SPARQL aside of CQELS. The system supports reasoning even in incomplete information cases through NAF, but like StreamRule it does not support historical data management. The SPARQL support of NAF was utilized in *StreamJess* to complement the difficulties for enabling NAF in Jess. Moreover, the reasoning results are returned as a JSON object to the corresponding web socket clients, while in *StreamJess* the reasoning results are returned as standard RDF data populating corresponding ontology classes. Their stream reasoning component is tested only with small amounts of input data. Our initial experimental results on *StreamJess* show better performance than the OClingo component implemented in StreamRule and (Ali et al., 2016) for small inputs, while system's performance evaluation for larger inputs is part of our future works. Jess is memory-intensive application, but recent Java Virtual Machines include flexible and configurable garbage collection subsystem which is responsible for finding and deleting unused objects (Hill, 2003). As argued by Hill (2003), the adjustment of two parameters: heap size and the object nursery size, has resulted with an improved 25% better performance.

Rscale (Liebig and Opitz, 2011) is another industrially-approved reasoning system which utilizes OWL 2 RL language profile to infer new knowledge. It enables incremental reasoning, non-monotonic and closed-world reasoning through translation of facts and rules into SQL tables and queries respectively. However, it does not support time-aware reasoning.

ETALIS (Anicic et al., 2010) together with EP-SPARQL (Anicic et al., 2011) enables CEP with stream reasoning. Even though ETALIS offers reasoning on time and location spaces it does not implement the windows feature. Time-based windows are supported through its wrapper EP-SPARQL, but complicated aggregations within windows are not supported (Le-Phuoc et al., 2012). Moreover, there is no support for triple-based windows too.

### 6.2. Semantic Web approaches

In the literature this approach is also referred to as interaction of ontologies and rules with tight semantic integration (Eiter et al., 2006). Even though the tight coupling of the model the rule language has distinct advantages e.g. no mapping mechanism is required between them, these approaches mainly suffer from limited expressiveness or decidability (Eiter et al., 2006). Thus, to date, there is not a tight-coupled approach which supports all the stream reasoning requirements. Approaches described by Keßler et al. (2009) and Wei and Barnaghi (2009) do not make any distinction between stream and static data, while also lack implementation. They prove that SWRL can be used to infer new and approximate knowledge in stream data domains. However, their approach does not consider time-aware and non-monotonic reasoning. Recently, a SPARQL extension (Anderson et al., 2016) that uses CONSTRUCT/WHERE clauses to express rules has been proposed. Yet again this approach does not consider non-monotonic reasoning. The works presented by Albeladi et al. (2015) Tallevi-Diotallevi et al. (2013) describe a Rete-based (Forgy, 1982) approach of RDFS entailment rules for producing data in a continuous manner. Although supporting time-aware and incremental reasoning, the approach does not deal with non-monotonic and closed-world reasoning. JNOMO (Calero et al., 2012) shows how SWRL can be extended to embrace non-monotonicity, CWA and NAF. Namely, NotExist operator is defined to "close" the world and to enable fact retraction. However, it does not deal with stream data, while inclusion of temporal reasoning is envisioned as per future works.

## 7 Conclusion

Until recently most of the SR research has been dedicated on ontology and query processing developments. Dealing with Big Data issues through query processing is not enough. Our work goes beyond the query processing achievements and thus focusing on rule level implications of stream data. SWRL, on its own, lacks the required expressivity level to reason over stream data. As a result, we built *StreamJess*, a production rule system capable of expressive reasoning over stream data. It layers Jess on top of C-SPARQL to enable time-aware, closed-world and non-monotonic reasoning on stream data domains. Jess and Jess Tab functions were used to enable non-monotonic reasoning. The system was validated in a WQM case study by running multiple C-SPARQL queries and Jess rules at the same time over the same RDF streams. Example 1 demonstrated how C-SPARQL queries can be used to filter RDF streams in time windows. The outputted results were processed by Jess rules to classify individual pH observations into appropriate WFD statuses. Furthermore, an investigation rule fired in case of critical status detection and identified the potential sources of pollution. Example 2 illustrated how WFD classification can be realized based on the average value of the observations. Except filtering the RDF streams were aggregated and then grouped by measurement site to classify and investigate $BOD_5$ observations.

The INWS ontology was also extended to meet the requirements for investigating sources of pollution. We believe that maintaining materializations on rules following ontology changes do not differ for stream data domains. However, a deeper research in this direction remains per future work. We also plan to compare *StreamJess* against the envisioned pure Semantic Web system (Jajaga and Ahmedi, 2015). Furthermore, our future work also includes enabling temporal operators (serial, sequence, etc.) on *StreamJess*.

## References

Ahmedi, L., Jajaga, E. and Ahmedi, F. (2013), 'An ontology framework for water quality management' in *Corcho, Ó., Henson, C. A. and Barnaghi, P. M. ed., SSN@ISWC*, Sydney, pp. 35-50.

Ahmedi, L., Sejdiu, B., Bytyçi, E. and Ahmedi, F. (2015), 'An Integrated Web Portal for Water Quality Monitoring through Wireless Sensor Networks', *International Journal of Web Portals (IJWP)*, Vol. 7 No. 1, pp. 28-46.

Albeladi, R., Martinez, K. and Gibbins, N. (2015), 'Incremental rule-based reasoning over RDF streams: An expression of interest', in *RDF Stream Processing Workshop at the 12th Extended Semantic Web Conference*, Portoroz, Slovenia.

Anicic, D., Fodor, P., Rudolph S., and Stojanovic, N. (2011), 'EP-SPARQL: a unified language for event processing and stream reasoning' in *WWW 2011*, pp. 635–644.

Barbieri, D. F., Braga, D., Ceri, S., Della Valle, E. and Grossniklaus, M. (2010), 'C-SPARQL: a continuous query language for RDF data streams', *International Journal of Semantic Computing*, Vol. 04 No. 01, pp. 3–25.

Barbieri, D. F., Braga, D., Ceri, S., Della Valle, E. and Grossniklaus, M. (2010) 'Incremental reasoning on streams and rich background knowledge', in *Proceedings of the Extended Semantic Web Conf. (ESWC 2010)*, Heraklion, Crete, Greece, pp.1-15.

*Basic Geo (WGS84 lat/long) Vocabulary*. [online] http://www.w3.org/2003/01/geo/. (Accessed 7 June 2016).

Boley, H., Kifer, M., Pătrânjan, P.-L. and Polleres, A. (2007), 'Rule interchange on the web', in *Reasoning Web*, LNCS, Springer, Heidelberg, Vol. 4636, pp. 269–309.

Calero, J. M. A., Ortega, A. M., Perez, G. M., Blaya, J. A. B. and Skarmeta, A. F. G. (2012), 'A non-monotonic expressiveness extension on the semantic web rule language', *Journal of Web Engineering*, Vol. 11 No. 2, pp. 93–118.

Compton, M., Barnaghi, P., Bermudez, L., GarcíaCastro, R., Corcho, O., Cox, S., Graybeal, J., Hauswirth, M., Henson, C. A., Herzog, A., Huang, V. A., Janowicz, K., Kelsey, W. D., Phuoc, D. L., Lefort, L., Leggieri, M., Neuhaus, H., Nikolov, A., Page, K. R., Passant, A., Sheth, A. P. and Taylor, K. (2012), 'The SSN ontology of the W3C semantic sensor network incubator group', *Journal of Web Semantics*, Vol. 17, pp. 25–32.

Della Valle, E., Ceri, S., Barbieri, D. F., Braga, D. and Campi, A. (2008), 'A first step towards stream reasoning' in *Proceedings of Future Internet Symposium (FIS 08)*, Springer, pp. 72–81.

Della Valle, E., Ceri, S., van Harmelen, F. and Fensel, D. (2009), 'It's a streaming world! Reasoning upon rapidly changing information', in *IEEE Intelligent Systems*, Vol. 24 No. 6, pp. 83–89.

Environment Agency. (2011), *Method statement for the classification of surface water bodies, v2.0 (external release)* [online], Monitoring Strategy v2.0, July 2011. (Accessed 7 June 2016).

Ermert, L. (2009), *Comparing Jess and Esper for Event Stream Processing*. Bachelor Thesis, Faculty IV - department computer science, Fachhochschule Hannover, Germany.

European Parliament and the Council of Europe. (2000) *Directive 2000/60/EC of 23 October 2000 establishing a framework for Community action in the Field of water quality*, O.J. L327/1.

Forgy, C. L. (1982), 'Rete: A fast algorithm for the many pattern/many object pattern match problem', *Artificial Intelligence*, Vol. 19, No. 1, pp. 17 – 37.

Foundation for Water Research. (2005) *Sources of Pollution* [online], Information Note FWR-WFD16. https://www.pik-potsdam.de/news/public-events/archiv/alter-net/former-ss/2007/03-09.2007/straskrabova/literature/Pollutionsourcs-WFD16-0.pdf. (Accessed 7 June 2016)

Gebser, M., Grote, T., Kaminski, R., Obermeier, P., Sabuncu, O. and Schaub, T. (2013), 'Answer set programming for stream reasoning', in *CoRR*.

Grosof, B. N., Gandhe, M. D. and Finin, T. W. (2002), 'SweetJess: Translating DamlRuleML to Jess', in: *Proceedings of International Workshop on Rule Markup Languages for Business Rules on the Semantic Web*, held at 1st International Semantic Web Conference.

Gruber, T. R. (1993), 'A translation approach to portable ontologies', in *Knowledge Acquisition*, Vol. 5 No. 2, pp. 199–220.

Hill, E. F. (2003), *Jess in action: Java rule-based systems*, Manning Publications Co., Greenwich, CT.

Horridge M. and Bechhofer, S. (2009), 'The OWL API: A Java API for working with OWL 2 ontologies', in *6th OWL Experienced and Directions Workshop*, Chantilly, Virginia.

Horrocks, I., Patel-Schneider, P. F., Boley, H., Tabet, S., Grosof, B. and Dean, M. (2004), *SWRL: A Semantic Web rule language combining OWL and RuleML*. [online] https://www.w3.org/Submission/SWRL/ (Accessed 7 June 2016)

*InWaterSense project.* [online] http://inwatersense.uni-pr.edu/ (Accessed 7 June 2016).

*INWS core ontology*. [online] http://inwatersense.uni-pr.edu/ontologies/inws-core.owl (Accessed 7 June 2016).

*INWS pollutants ontology*. [online] http://inwatersense.uni-pr.edu/ontologies/inws-pollutants.owl (Accessed 7 June 2016).

*INWS regulations ontology*. [online] http://inwatersense.uni-pr.edu/ontologies/inws-regulations.owl (Accessed 7 June 2016).

Jajaga, E., Ahmedi, L. and Abazi-Bexheti, L. (2013), 'Semantic Web trends on reasoning over sensor data' in *8th South East European Doctoral Student Conference*, Greece, pp. 284-293.

Jajaga, E., Ahmedi, L. and Ahmedi, F. (2015), 'An Expert System for Water Quality Monitoring Based on Ontology', in *MTSR 2015: Proceedings of the 9th Metadata and Semantics Research Conference*, Manchester, UK, Vol. 544 pp. 89-100.

*Jess Wiki: Jess Tab*. [online] http://www.jessrules.com/jesswiki/view?JessTab (Accessed 7 June 2016).

Keßler, C., Raubal M. and Wosniok, C. (2009), 'Semantic rules for context-aware geographical information retrieval', in *EuroSSC 2009*, LNCS, Springer, Vol. 5741, pp. 77–92.

Lanzanasto, N., Komazec, S. and Toma, I. (2009), *Deliverable D4.8: Reasoning over real time data streams*, ENVISION Consortium.

Le-Phuoc, D., Dao-Tran, M., Pham, M.-D., Boncz, P., Eiter, T. and Fink, M. (2012), 'Linked stream data processing engines: facts and figures', in *The Semantic Web–ISWC 2012*, Springer, pp. 300–312.

Le-Phuoc, D., Dao-Tran, M., Xavier Parreira, J. and Hauswirth, M. (2011) 'A native and adaptive approach for unified processing of linked streams and linked data', in *The International Semantic Web Conference –ISWC 2011*, pp. 370–388.

Liebig, T. and Opitz, M. (2011), 'Reasoning over dynamic data in expressive knowledge bases with Rscale', in *The 10th International Semantic Web Conference*, Bonn, Germany.

Margara, A., Urbani, J., van Harmelen, F. and Bal, H. (2014), 'Streaming the web: reasoning over dynamic data' in *Web Semantics: Science, Services and Agents on the World Wide Web*, 25(0), pp. 24–44.

McBride, B. (2004), 'Jena: implementing the RDF model and syntax specification', in *Proceedings at Semantic Web Workshop (WWW)*.

Mileo, A., Abdelrahman, A., Policarpio, S. and Hauswirth, M. (2013) 'StreamRule: a nonmonotonic stream reasoning system for the semantic web', in *RR 2013*, LNCS, Springer, Heidelberg, Vol. 7994, pp. 247–252.

O'Connor M. J. and Das, A. K. (2009) 'SQWRL: a query language for OWL', in *OWL: Experiences and Directions (OWLED), 6th International Workshop*, Chantilly, VA.

O'Connor, M. J., Knublauch, H., Tu, S. W., Grossof, B., Dean, M., Grosso, W. E. and Musen, M. A. (2005) 'Supporting rule system interoperability on the Semantic Web with SWRL', in *4th International Semantic Web Conference*, Galway, Ireland, Springer Verlag, LNCS Vol. 3729, pp. 974-986.

OWL time ontology. [online] http://www.w3.org/TR/owl-time/ (Accessed 7 June 2016)

Statutory Instruments. (2009), *European Communities Environmental Objectives (Surface Waters) Regulations 2015* [online], S.I. No. 386 of 2015. http://www.irishstatutebook.ie/eli/2015/si/386/made/en/pdf. (Accessed 7 June 2016)

Tallevi-Diotallevi, S., Kotoulas, S., Foschini, L., Lecue F. and Corradi (2013), 'A Real-time urban monitoring in Dublin using semantic and stream technologies', in *The Semantic Web ISWC 2013*, Vol. 8219 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, pp. 178–194.

Volz, R., Staab, S. and Motik, B. (2005), 'Incrementally maintaining materializations of ontologies stored in logic databases', *Journal of Data Semantics II*, Vol. 3360, pp. 1–34.

Wang E. and Kim, Y. S. (2006), 'A teaching strategies engine using translation from SWRL to Jess', in *8th International Conference on Intelligent Tutoring Systems*, June 26-30 2006, LNCS Vol. 4053, pp. 51-60.

Wei W. and Barnaghi, P. (2009), 'Semantic annotation and reasoning for sensor data', in *Smart Sensing and Context*, pp.66-76.

Zaniolo, C. (2012), 'Logical foundations of continuous query languages for data streams' in *Proceedings of Datalog*, pp. 177–189.

Advancing Discovery in Science and Engineering. Computing Community Consortium. Spring 2011.

Luckham, D. and Roy Schulte, W. (2011), Event Processing Glossary -Version 2.0. Event Processing Technical Society, 2nd edition.

Eiter, T., Ianni, G., Polleres, A., Schindlauer, R. and Tompits, H. (2006), 'Reasoning with Rules and Ontologies', in *Reasoning Web*, Second International Summer School 2006, Tutorial Lectures, LNCS, vol. 4126, Springer, pp. 93–127.

Sottara, D., Bragaglia, S., Mello, P., Pulcini, D., Luccarini, L., and Giunchi, D. (2012), 'Ontologies, Rules, Workflow and Predictive Models: Knowledge Assets for an EDSS, in

*International Environmental Modelling and Software Society (iEMSs)*, 2012 International Congress on Environmental Modelling and Software Managing Resources of a Limited Planet, Sixth Biennial Meeting, Leipzig, Germany.

Chau, K. W. (2007), 'An ontology-based knowledge management system for flow and water quality modeling' in Advances in Engineering Software, Vol. 38, no. 3, pp. 172-181.

Walzer, K., Groch, M., and Breddin, T. (2008), 'Time to the Rescue – Supporting Temporal Reasoning in the Rete Algorithm for Complex Event Processing' in *Proceedings of 19th Int. Conf. on Database and Expert Systems Applications*, Springer-Verlag, pp. 635–642.

Komazec, S., and Cerri, D. (2011), 'Towards Efficient Schema-Enhanced Pattern Matching over RDF Data Streams' in *10th ISWC*, Springer, Bonn, Germany.

Schmidt, K., Stuhmer, R., and Stojanovic, L. (2008), 'Blending complex event processing with the rete algorithm' in *1st International workshop on Complex Event Processing for the Future Internet colocated with the Future Internet Symposium*, CEUR Workshop Proceedings, Vol. 412.

Hardy, C. E. (2013), *Stream Reasoning on Resource-Limited Devices*. University of Dublin, Dublin, United Kingdom.

Ali, M. I., Ono, N., Kaysar, M., Shamszaman, Z. U., Pham, T.-L., Gao, F., Griffin, K., and Mileo, A. (2016) 'Real-time Data Analytics and Event Detection for IoT-enabled Communication Systems', *Journal of Web Semantics: Science, Services and Agents on the World Wide Web*.

Anicic, D., Fodor, P., Rudolph, S., Stuhmer, R., Stojanovic, N., Studer, R. (2010), 'A Rule-Based Language for Complex Event Processing Reasoning', in *Proceedings of the Fourth International Conference on Web reasoning and rule systems*, pp. 42-57, Springer-Verlag Berlin, Heidelberg.

Anderson, J., Athan T., and Paschke, A. (2016), 'Rules and RDF Streams' - A Position Paper, in *Proceedings of the RuleML 2016 Challenge*, Doctoral Consortium and Industry Track hosted by the 10th International Web Rule Symposium (RuleML 2016), New York, USA.