Enabling Knowledge Management in Complex Industrial Processes Using Semantic Web Technology

Katarina Milenković[1], Simon Mayer[2,3], Konrad Diwold[1,4] and Josef Zehetner[5]
[1]Pro²Future GmbH, 8010 Graz, Austria
[2]University of St. Gallen, CH-9000 St.Gallen, Switzerland
[3]ETH Zürich, 8092 Zürich, Switzerland
[4]TU-Graz, 8010 Graz, Austria
[5]AVL List GmbH, 8020 Graz, Austria
katarina.milenkovic@pro2future.at
simon.mayer@unisg.ch
konrad.diwold@pro2future.at
josef.zehetner@avl.com

Abstract: Complex industrial processes produce a multitude of information during the product/service lifecycle. Those data are often stored, but rarely used in the context of overall process optimization, due to their unstructured format and the inability to integrate them with stored formal knowledge about the domain. This paper proposes a way to mitigate this problem, by extending the standard SPARQL query language to enable the integration of formal knowledge and unstructured data, as well as their joint processing. The paper constitutes an initial definition of the proposed SPARQL extension and demonstrates its applicability in the context of selected examples.

Keywords: knowledge management, semantic web, metadata, SPARQL

1 Introduction

Whether they are manufacturing physical products or delivering virtual services, corporations engage in a wide variety of processes that are interrelated within and across different phases along the product/service lifecycle. In addition to achieving the core functional purpose within each step along the lifecycle (e.g., manufacturing, testing, maintenance, etc.), each process yields information about the product as well as about the process itself. There are countless examples of such information such as: data about the *quality* of the manufactured product in various production stages; data about planned, scheduled, and executed *testing* procedures; performance/measurement data of the machines involved in the production process; etc. Corporations have started to collect and store this information to improve their processes, as they are expecting that the collected data might enable them to extract valuable insights about the product and the production

process, thus enabling them to improve individual process steps or individualizing the product/service for customers.

Recently a number of data-based approaches for industrial processes monitoring have been brought forward. These approaches use different methods for managing and post-processing process data (e.g., Manabu and Yoshiaki 2008; Zhiqiang et al. 2013). This article proposes a novel approach to link (and thus utilize unstructured) process data generated within the product/service lifecycle with structured process information. This is achieved by utilizing semantic web technologies, in particular ontologies, by focusing on one important challenge in the context of complex industrial processes: the development of methods that enable the *simultaneous processing and exploitation of structured and unstructured data*. To this end, a solution to bridge the gap between the two types of data is proposed.

In order to manage structured, semantically enriched data, the Resource Description Framework (RDF; Brickley and Guha 2014) is used. RDF is a specification by the World Wide Web Consortium (W3C), for the standardization of Semantic Web technologies. Within RDF, relationships between objects (henceforth "resources") are described using <subject - predicate - object> triples, with the predicate constituting the relationship between a subject and an object (e.g., "Apple is a Fruit"). Triples can be combined into directed RDF graphs which can be queried via query languages such as SPARQL (Prud'hommeaux et al. 2013) to retrieve context information.

Although RDF is very convenient for storing structured data, it is not appropriate for storing unstructured product lifecycle data (Hu et al. 2016; Cudré-Mauroux et al. 2013). This is due to the fact that in addition to containing large amounts of data (such as time series), unstructured data (as their name suggests) do not follow any explicit standard data models. Storing and querying such data with standard tools in the semantic technologies domain (such as RDF and SPARQL) is very inefficient and does not scale (Bornea et al. 2013). At the same time however, the very nature of these data suggests that individual values do not provide much information, but make sense (and could add value to the overall process) only if viewed in context with other values (e.g., parameters of another involved machine that were recorded at the same time period).

To link structured and unstructured data here an extension of the SPARQL query language is proposed. This extension includes several new operators, which integrate querying for structured and unstructured data into SPARQL. To accomplish this, we make use of another method from the Semantic Web domain: Linked Data (Bizer et al. 2008), i.e., a way of publishing data so that it can be easily interlinked. The idea is to store unstructured data so that they can be accessed through a Web service and to store only a link to that service inside the RDF graph. New SPARQL operators then enable retrieving unstructured data from a given link and processing retrieved results in an enclosing SPARQL query/queries. This allows to host each data within its respective habitat (and thus prevents performance and scaling problems), while still enabling to access and integrate the data.

The rest of the paper is organized as follows. *Section 2* introduces the problem and gives further motivation for our work. *Section 3* discusses related work. *Section 4* presents a novel solution for integrating structured and unstructured data. *Sections 5* and *6* describe all necessary changes applied to the SPARQL grammar and algebra, to enable the introduction of new operators. *Section 7* concludes the article.

2 Problem description and technological background

Information created and emitted by processes during the product/service lifecycle is usually designed for a specific context. This commonly entails that information is stored in formats that lack proper facilities for preserving metadata along with it (e.g., as plain comma-separated-values inside CSV files). This represents a major obstacle to a later analysis since it undermines the proper contextualization of the data. Colloquially, this way of storing data is referred to as “data lakes”: storage repositories that hold vast amounts of raw data in its native format until needed and without annotations that would support later contextualization (Anadiotis 2017). The amount of data stored in this way is constantly increasing, which (by itself) is starting to hinder efficient data analysis, as corporations are dealing with unstructured files of sizes in the range of Giga- or even Terabytes that cannot anymore be processed efficiently in practice.

To enable the efficient interpretation and extraction of useful, actionable information from such data, it is necessary to store it in a way, that ensures that relevant metadata (e.g., the data acquisition context and provenance information) is preserved and can be automatically processed. This is already relevant when attempting to optimize individual processes, for instance by enabling customized testing processes that are optimized for an individual product and its context. Additionally, preserving metadata is crucial for the analysis of systems of interrelated processes – especially if these processes are related to different phases in the lifecycle of a product or service. In this case, metadata is required to construct contextual bridges across the product lifecycle phases to answer questions such as: “How do quality variations during production translate to issues during the in-use phase of a product?” or ”What implications do product usage patterns have for the design of the product itself, and for add-on services?”.

2.1 Open semantic framework

In order to create a system for managing and using semantically enriched data, the Open Semantic Framework (OSF; Mayer et al. 2017) was used. OSF provides an easy way of creating and managing groups of ontologies (so-called *knowledge packs*, which are built on top of RDF), thus supporting the maintenance, curation, and access to stored structured data that is relevant to a specific problem. Knowledge packs formalize possessed knowledge and also include metadata (context information, definitions, categories, etc.) about concepts in the domain.

Within OSF, the querying of knowledge packs is achieved via an HTTP API, which is linked to SPARQL queries. The querying is controlled by enabling only pre-formulated query templates that are specific to the knowledge pack, or group of knowledge packs, at hand. This gives designers of knowledge packs the ability to plan what information can be retrieved from a particular knowledge pack and eases the integration of the OSF in superposed processes as the integration can be achieved via well-defined HTTP calls. It also gives corporations the ability to price knowledge access based on the power/value of the executed SPARQL templates. Besides standard selection queries, OSF also supports update and construct queries. This allows a dynamic extension of knowledge packs during operation by integrating new data in the system, thus creating the possibility for the system (and its underlying knowledge packs) to incorporate new information, thereby “learning” about new resources and relationships along the way. OSF is built on top of Apache Jena (McBride 2002) to manage RDF graphs and execute SPARQL queries over them.

To perform a SPARQL query, a sequence of steps needs to be executed. The key steps in this process are:

- *Query Parsing:* analyzing string description of the query according to the given formal grammar rules;
- *Algebra Generation:* generating an algebraic expression for the parsed query from the previous step;
- *Query Evaluation:* executing the created algebraic expression and producing a solution.

Every extension of the existing SPARQL query language that intends to be widely accepted in the community should be in accordance with its already well-known syntax. New extensions should be compatible with the old versions, adding new possibilities while not obstructing SPARQL's core functionality.

2.2 Time series databases

While unstructured data can be stored in a variety of formats, it usually misses relevant metadata that would enable its integration in the knowledge management system. Additionally, as there is usually a large amount of those data, RDF is not a suitable means of storage.

As in the context of complex industrial processes one usually refers to a collection of data (values) generated during different phases of the product/service lifecycle, NoSQL time-series databases seem like an ideal candidate to store this information. Time-series databases are a type of database that are optimized for handling time-series data (i.e. a series of values (data points) indexed in time order). In our system, we used the popular InfluxDB time-series database. Optimized for storage and retrieval of time-series data, InfluxDB was ranked number one in DB-Engines Ranking of Time Series DBMS list in January 2016 (https://db-engines.com/en/ranking/time+series+dbms accessed on 05.03.2019.), and has maintained its ranking since. It provides support for a different kind of data analysis and querying using a SQL-like query language called InfluxQL. Another advantage of InfluxDB for our purposes is that it possesses an HTTP API which enables easy interaction with the database through its endpoints. The */query* endpoint is used to query data and manage databases. It accepts both GET and POST HTTP requests and returns the response in JSON format. These characteristics make it very suitable as a starting point to integrated structured knowledge with unstructured time-series data.

3 Related work

Accessing data through hyperlinks is one of the basic elements of the Semantic Web. The idea of storing merely a link to unstructured time-series data inside an RDF graph was proposed as part of the Melodies project (Almolda 2016), in the context of processing hydrological time-series data. Almolda (2016) used RDF to store data about the underlying model and provided hyperlinks to access the connected time-series data which were stored in relational databases. A series of Web services was implemented in order to enable access to time-series data through embedded hyperlinks. Those embedded hyperlinks already had to include all the necessary parameters to retrieve desired time-series data. It was not possible to add parameters afterward. This inability reduced users flexibility with respect to

accessing remote data and allowed the use of only pre-defined hyperlinks. While Almolda (2016) integrated the two types of data, he used the official SPARQL query language to query RDF triples. This enabled him to manage embedded hyperlinks inside those queries, but not to manage and process time-series data themselves. We instead chose to directly extend SPARQL in order to provide integrated tooling for processing the two types of data we handle.

Other researchers and practitioners have proposed to extend SPARQL with additional features as well. Lefrançois M. et al. (2017) created SPARQL-Generate, an extension of SPARQL that generates RDF from RDF dataset, and a set of documents in arbitrary formats. Mizell et al. (2014) added graph functions to enable SPARQL-based graph analytics. They included a small set of build-in graph function.

A flexible extension of SPARQL, named f-SPARQL is proposed by Cheng J. et al. (2010). Their extension introduces a fuzzy logic into SPARQL. Bolles et al. (2008) extended SPARQL with the possibility to process data streams.

Historically, several proposed extensions to SPARQL have proven to be very useful and are today included as official SPARQL query language extensions. This includes Angles and Gutierrez (2011) who proposed including subqueries into SPARQL, and Alkhateeb F. et al. (2007) who developed the PSPARQL query language, an extension of the SPARQL query language which allows querying RDF triples using graph patterns whose predicates are regular expressions. Another popular and widely excepted SPARQL extension is C-SPARQL proposed by Barbieri et al. (2010). C-SPARQL adds support for executing continuous queries over RDF data streams.

4 Integrating structured and unstructured data

This section presents a way to link structured and unstructured data by the appropriate web-based embedding of unstructured formats which enable the usage of this data in the context of semantic technology platforms such as OSF. Specifically, the Web-based nature of OSF enables the use of embedded hyperlinks to access unstructured databases, thereby enabling us to integrate these two types of information while not obstructing processes that depend on the raw unstructured data. In other words, storing links to time-series data in RDF graphs, not the data themself will be enough to later retrieve those data. Table 1 shows part of an ontology that describes one complex industrial development processes.

We propose using semantic relationships to enable incorporation of unstructured information (e.g., quality measures of a production process) in the process, thereby dynamically expanding the existing process knowledge and improving the expressiveness of the system and therefore the reasoning that can be achieved. The main challenge to enable this are the limitations of the current SPARQL syntax. We hence propose to extend the SPARQL query language by adding new operators that enable retrieving time-series data through hyperlinks that are embedded in the ontologies (by sending appropriate HTTP requests and parsing the obtained responses), and further processing retrieved data in other SPARQL queries. The extension to the SPARQL query language is demonstrated by implementing several changes in the SPARQL processor for Apache Jena. In the following, we

describe in greater detail all the necessary extensions and changes made to enable the usage of SPARQL to accomplish this integration. We strive to keep the established query execution process unchanged as far as possible. In order to be able to properly detect new symbols and replace them with corresponding operators of algebraic expressions, we extended the SPARQL grammar and defined new algebraic operators. The rest of the steps of the query execution, i.e. evaluation, were merely extended with facilities to handle the additional operators but are otherwise equal to regular SPARQL executions. Throughout the process, our work remains compatible with the W3C SPARQL definition.

Table 1: Part of an ontology describing one complex industrial development processes

```
@prefix cavl:    <http://pro2future.at/schemas/cavl#> .
@prefix owl:     <http://www.w3.org/2002/07/owl#> .
@prefix rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs:    <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xsd:     <http://www.w3.org/2001/XMLSchema#> .
...
cavl:DevTaskProcess
        rdf:type owl:Class ;
        rdfs:comment "Dev Task Process class that models testing process."^^xsd:string ;
        rdfs:label "Dev Task Process"^^xsd:string ;
        cavl:devTaskSetup cavl:DevTaskSetup ;
        cavl:targetCharacteristic cavl:CharacteristicValue ;
        cavl:linkToTimeSeries "http://192.168.56.104:8086/query"^^xsd:string ;
.
...
```

5 Extended SPARQL grammar

The first step towards extending SPARQL by adding new operators is to extend the SPARQL grammar in order to be able to parse queries containing newly inserted elements. This extension should not affect the existing, already well-known SPARQL operators. While extending the grammar, we tried to preserve the logic of SPARQL syntax as much as possible. The extension fits well into existing SPARQL syntax and is a natural addition to already existing possibilities of the language. Table 2 shows the extension of the EBNF notation used in the grammar (based on Prud'hommeaux et al. 2013). Newly added terms are shown in bold font.

Table 2: Extension of the EBNF notation

Query	::=	Prologue (SelectQuery | **TimeseriesQuery** | ConstructQuery | DescribeQuery | AskQuery) ValuesClause
TimeseriesQuery	::=	**TimeseriesClause** **HyperlinkClause** **ParametersClause*** **TimeseriesModifier**
TimeseriesClause	::=	'**TIMESERIES**' **TimeseriesVar**+
TimeseriesVar	::=	'(' **VAR3** 'AS' Var ')'
VAR3	::=	'**%**' **JSONDATANAME**
JSONDATANAME	::=	(PN_CHARS_U | [0-9]) (PN_CHARS_U | [0-9] | #x00B7 | [#x0300-#x036F] | [#x203F-#x2040])* (**PN_CHARS_I**)?

PN_CHARS_BASE	::=	[A-Z] | [a-z] | [#x00C0-#x00D6] | [#x00D8-#x00F6] | [#x00F8-#x02FF] | [#x0370-#x037D] | [#x037F-#x1FFF] | [#x200C-#x200D] | [#x2070-#x218F] | [#x2C00-#x2FEF] | [#x3001-#xD7FF] | [#xF900-#xFDCF] | [#xFDF0-#xFFFD] | [#x10000-#xEFFFF]
PN_CHARS_U	::=	PN_CHARS_BASE | '_'
PN_CHARS_I	::=	#x002E **JSONDATANAME** | #x005B [0-9]* #x005D (PN_CHARS_I)?
Var	::=	VAR1 | VAR2
HyperlinkClause	::=	'**HYPERLINK**' (IRIREF |**SelectHyperlinkQuery**)
SelectHyperlinkQuery	::=	'{' '**GETLINK**' Var DatasetClause* WhereClause OrderClause? '}'
DatasetClause	::=	'FROM' (DefaultGraphClause | NamedGraphClause)
WhereClause	::=	'WHERE'? GroupGraphPattern
OrderClause	::=	'ORDER' 'BY' OrderCondition+
IRIREF	::=	'<' ([^<>"{}|^`\]-[#x00-#x20])* '>'
ParametersClause	::=	'**PARAMETER**' **VAR4**
VAR4	::=	**PARAM_NAME** **PARAM_VALUE**
PARAM_NAME	::=	'**$**' **PARAMETER_NAME**
PARAM_VALUE	::=	'"' **PARAMETER_VALUE** '"'
PARAMETER_NAME	::=	([#x0000-#x009F] | [#x00A1-#xEFFFF])*
PARAMETER_VALUE	::=	([#x0000-#x009F] | [#x00A1-#xEFFFF])*
TimeseriesModifier	::=	**TimeseriesOrderClause**? LimitOffsetClauses?
TimeseriesOrderClause	::=	'ORDER' 'BY' **TimeseriesOrderCondition**+
TimeseriesOrderCondition	::=	(('ASC' | 'DESC') '(' Var ')') | Var
SelectQuery	::=	SelectClause DatasetClause* WhereClause SolutionModifier
WhereClause	::=	'WHERE'? GroupGraphPattern
LimitOffsetClauses	::=	LimitClause OffsetClause? | OffsetClause LimitClause?
GroupGraphPattern	::=	'{' (SubSelect | **SubTimeseries** | GroupGraphPatternSub) '}'
SubTimeseries	::=	**TimeseriesClause** **HyperlinkSubClause** **ParametersClause*** **TimeseriesModifier**
HyperlinkSubClause	::=	'**HYPERLINK**' (IRIREF | **SelectHyperlinkSubQuery**)
SelectHyperlinkSubQuery	::=	'{' '**GETLINK**' Var WhereClause OrderClause? '}'

The most important change introduced by our extension is the creation of a new type of query, the TIMESERIES query -- marked as *TimeseriesQuery* in Table 2. This type of query enables retrievement of data from a given link, processing those data and its interpretation as a SPARQL result (result has the same form as a result of regular SELECT query). That enables post-processing of retrieved data within an enclosing SELECT query. The syntax of a *TimeseriesQuery* syntax is similar to the syntax of a regular SELECT query (see Table 3).

The *TimeseriesClause* element refers to the result bindings of the query. Accessing a Web service with time-series data through a link given in TIMESERIES query (see below: *HyperlinkClause* and *ParametersClause* elements) returns a response in JSON format -- an array of JSON responses. *TimeseriesQuery* has to interpret that response in a well-known SPARQL form in order to enable its further use in the enclosing queries. *TimeseriesClause* is what supports this process: it defines parts of JSON responses which will be present in TIMESERIES query result bindings in a placeholder variable referred to as *VAR3* (note that the SPARQL *SelectClause* already defines *VAR1* and *VAR2*), and its corresponding naming in *Var*. As JSON responses might not contain only simple key-string/number/boolean/null value pairs, but also complex key-JSON object value and/or key-JSON array value pairs, *VAR3* enables access to simple values to an arbitrary level of nesting inside JSON response. We use the typical JSON notation to express this, i.e. a dot followed by a field name to access a JSON field value and an index value within square brackets to access an element of a JSON array.

The *HyperlinkClause* element defines a hyperlink to a Web service with data. The hyperlink can be given as an Internationalized Resource Identifier reference (*IRIREF*), or as a result of a nested subquery called a *SelectHyperlinkQuery*. The possibility to retrieve a link as a result of a nested query enables users to embed unstructured data into the knowledge management system, i.e. store a hyperlink to a Web service with time-series data as a part of an ontology (subject or object in an RDF triple). *SelectHyperlinkQuery* has a very similar form to a regular SELECT query; it however makes use of the GETLINK keyword, can have only one column in its result bindings, and supports only a single returned result.

The *ParametersClause* element defines parameters to be appended to the hyperlink defined in *HyperlinkClause* when accessing remote data. One can set a desired number of parameters. Every parameter is set using the keyword PARAMETER after which follow the pair of parameter name, parameter value. Importantly in the context of using InfluxDB to store unstructured data with these new SPARQL options, *ParametersClause* enables creating InfluxQL queries as a part of a hyperlink directly using query parameters. *ParameterClause* is an optional element of *TimeseriesQuery*.

TimeseriesModifier element is analogous to standard SPARQL *SolutionModifier*, but with a narrowed scope that only permits *order* and *limit* modifiers.

In order to enable post-processing of result bindings of TIMESERIES query, we created the possibility of nesting time-series queries inside SELECT queries. This type of subquery is referred to as *SubTimeseries*. Most of the elements of *SubTimeseries* are the same as for *TimeseriesQuery*, with the exception that *SubTimeseries* does not have a *DatasetClause* -- this is similar in name and function to SPARQL's *SubSelect*, which also misses a *DatasetClause*.

5.1 An example of combining structured and unstructured data using our SPARQL extension

Let's observe the case when we manage information about a complex development process in a corporation. Every complex development process can be viewed as a set of different development tasks. Each of those tasks is used during the development to determine some specific characteristic of the process. The task is being performed as a sequence of stages, where that sequence is not strictly defined within the task and stages can be repeated.

During development processes, different tasks are being executed and many useful information about the process is produced. Those data could give insights into how different characteristics change during development task, how different stages of the task affect the characteristic, etc. but their deficiency is their form (unstructured data), and lack of relevant metadata. We store them in an InfluxDB. Expert knowledge about the domain (structured data with relevant metadata) is stored in knowledge packs in RDF form. Knowledge packs contain information about different types of development tasks that can be performed, their setup, characteristics they determine, etc. They also contain links to Web service with the InfluxDB instance containing unstructured data about the domain.

In Table 3, we give a working example of using our SPARQL extension in described domain. This example demonstrates one simple enrichment of unstructured data by linking those data with the proper characteristic. We are querying for the information about all the stages that have to be executed in order to determine given characteristic (stage itself and the impact it has on the characteristic). Using the GETLINK operator, we retrieve a link to a Web service with InfluxDB instance. The PARAMETER operator enables us to set appropriate parameters for accessing InfluxDB instance: using parameter *$db* we set the target database for the query; using parameter *$q* we set InfluxQL string to execute. TIMESERIES query will combine retrieved link and provided parameters to form a complete hyperlink to access desired data.

Table 3: An example of a query using newly introduced operators

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX cavl: <http://pro2future.at/schemas/cavl#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>

SELECT ?reflectCharacteristic ?stage ?influence
WHERE {
   {
      TIMESERIES (%tags.stage AS ?stage)
                 (%values[0][1] AS ?influence)
      HYPERLINK {
         GETLINK ?link
         WHERE {
            ?devTaskProcessType rdfs:subClassOf ?devTaskProcess .
            ?devTaskProcessType cavl:targetCharacteristic ?characteristic .
            ?devTaskProcess cavl:linkToTimeSeries ?link .
         }
         ORDER BY DESC(?link)
      }
      PARAMETER $db "my_db"
      PARAMETER $q "SELECT *
                    FROM (
                       SELECT MEAN(\"difference\") as \"influence\"
                       FROM (
                          SELECT \"stage\",
                                 ABS(\"in_value\"-\"in_target\")/\"in_target\" -
                                    ABS(\"out_value\"-\"out_target\")/\"out_target\" as \"difference\"
                          FROM \"FuelConsumption\"
                       )
                       GROUP BY \"stage\"
                    )
```

```
                        GROUP BY \"stage\""
    ORDER BY ASC(?characteristicName) DESC(?influence)
    LIMIT 10
  }

  BIND (?characteristic as ?reflectCharacteristic) .
}
```

6 Extended SPARQL algebra

In addition to extending the SPARQL grammar, also its algebra needs to be augmented by precisely defining all new operators. As when extending SPARQL grammar, we tried not to deviate much from the well-known SPARQL algebra.

Translating SPARQL query objects to an algebraic expression is one of the most important steps when executing a query. As familiar from the standard, the process of generating algebra expression converts queries by first converting all of its nested queries, one by one. This process is applied recursively on all subqueries.

Each of the abstract query symbols has to have a defined operator for its evaluation. In the following, we define all newly inserted symbols and their operators, in the same way as this is done in the SPARQL W3C recommendation (Prud'hommeaux et al. 2013).

Since we introduced a new query type and its element *TimeseriesClause*, we need to define two new operators: *alias* and *extract*.

Definition: Alias

Let μ be a solution mapping, Ω a multiset of solution mappings, *var* a variable and *key* an identifier of a field of a JSON object. Every *key AS var* element in *TimeseriesClause* is being translated to *alias(Ω, var, key)* term in SPARQL algebra.

Then we define:

$$alias(\mu, var, key) = \mu \cup \{ (var,value) \mid var \text{ not in } dom(\mu) \text{ and } value = key(\mu) \},$$

alias(μ, var, key) = μ if *var* not in *dom(μ)* and *key(μ)* is an error,

alias(μ, var, key) is undefined when *var* in *dom(μ)*,

$$alias(\Omega, var, key) = \{ alias(\mu, var, key) \mid \mu \text{ in } \Omega \}.$$

The cardinality of solution mapping in a multiset of solution mappings remains the same after applying this operator, ie. $card[alias(\Omega, var, key)](\mu) = card[\Omega](\mu)$.

Definition: Extract

Let Ψ be a multiset of solution mappings and *PV* a set of variables. Every *TimeseriesClause* with *PV* set of variables is being translated to *extract(Ψ, PV)* term in SPARQL algebra. For mapping μ, *extract(μ, PV)* is the restriction of μ to variables in *PV*.

Then we define:

$$extract(\Psi, PV) = \{ extract(\mu, PV) \mid \mu \text{ in } \Psi \}.$$

The cardinality of solution mapping in a multiset of solution mappings remains the same after applying this operator, ie. *card[extract(Ψ, PV)](μ) = card[Ψ](μ)*.

The order of *extract(Ψ, PV)* must preserve any ordering given by *OrderBy*.

The core element of this extension is the *HyperlinkClause*, which contains a link for retrieving the remote data. That link can be given as a simple *IRIREF* or as a result of a nested subquery. The two different ways of setting the link imply the need for introducing two new operators since there is a need for different behavior in each of the cases. One operator (URI value) will be used whenever the link is given as an *IRIREF*. It will in that case simply combine the given link with parameters to form a complete link to access remote data. The other operator (URI subquery) will first retrieve link through a given subquery and then combine it with parameters to form a complete link to access remote data. It will be used when link is given as a result of nested subquery.

Definition: URI value

Let *uri_value* be the way to retrieve data from remote server *S*, *iri* a link to a Web service, *params* URI parameters for that link and let server *S* return a result in form of JSON array object. *μ* is a solution for *uri_value* from *S* when a response from *S* is a JSON array with keys that make a pattern instance mapping *P* and *μ* is the restriction of *P*.

Then we define:

Ω = *uri_value(iri, params)*, μ element of Ω.

The number of distinct *μ* in *Ω* equals the number of JSON elements in returned JSON array, ie. *card[Ω] = length(resulting JSON array)*.

Definition: URI subquery

Let *uri_subquery* be the way to retrieve data from remote server *S*, Ω_1 multisets of solution mappings containing a link to a Web service, *params* URI parameters for that link and let server *S* return a result in form of JSON array object. *μ* is a solution for *uri_subquery* from *S* when a response from *S* is a JSON array with keys that make a pattern instance mapping *P* and *μ* is the restriction of *P*.

Then we define:

Ω = *uri_subquery (iri in* Ω_1*, params)*, μ element of Ω.

If $card[\Omega_1]!=1$ and $card[\Omega_1](\mu_1)!=1$ is an error.

The number of distinct *μ* in *Ω* equals the number of JSON elements in returned JSON array, ie. *card[Ω] = length(resulting JSON array)*.

ParametersClause of *TimeseriesQuery* introduces the possibility for users to define an arbitrary number of parameters, which will be used later when accessing remote time-series data.

Definition: Parameter Pattern

A parameter pattern is a member of the set: (*parameter_name* x *parameter_value*).

Every *PARAMETER parameter_name parameter_value* from *ParameterClause* is being translated to *parameter parameter_name parameter_value* in SPARQL algebra.

Definition: Basic Parameter Pattern

A Basic Parameter Pattern (BPP) is a set of parameter patterns.

Any adjacent parameter patterns from *ParameterClause* collected together form a BPP. Let

PARAMETER parameter_name1 parameter_value1
PARAMETER parameter_name2 parameter_value2

be a *ParameterClause*, then it is being translated to

bpp(
(parameter parameter_name1 parameter_value1)
(parameter parameter_name2 parameter_value2))

in SPARQL algebra.

Table 4 shows how the SPARQL query from Table 3 is translated to SPARQL algebra.

Table 4: An example of how a query that makes use of our newly introduced operators is translated to SPARQL algebra.

```
(project (?reflectCharacteristic ?stage ?influence)
 (extend ((?reflectCharacteristic ?characteristic))
  (extract ( ?stage ?influence)
   (order ((asc ?characteristicName) (desc ?influence))
    (alias ((?influence ?values[0][1]))
     (alias ((?stage ?tags.stage))
      (uri_subquery
       (slice _ 1
        (project (?link)
         (order ((desc ?link))
          (bgp
           (triple ?devTaskProcessType <http://www.w3.org/1999/02/22-rdf-syntax-ns#subClassOf>
                                                                                    ?devTaskProcess)
           (triple ?devTaskProcessType <http://pro2future.at/schemas/cavl#targetCharacteristic>
                                                                                      ?characteristic)
           (triple ?devTaskProcess <http://pro2future.at/schemas/cavl#linkToDataseries> ?link)
          ))))
       (bpp
        (parameter $db "my_db")
        (parameter $q "SELECT *
                       FROM (
                         SELECT MEAN("difference") as "influence"
                         FROM (
                           SELECT "stage",
                                  ABS("in_value"-"in_target")/"in_target" -
                                                ABS("out_value"-"out_target")/"out_target" as "difference"
                           FROM "FuelConsumption"
```

```
                    ) GROUP BY "stage"
                ) GROUP BY "stage")")
))))))))
```

7 Conclusion

This paper described an extension of SPARQL query language we proposed as a solution for integrating structured and unstructured data. Structured data are semantically enriched data that contain relevant metadata. Those data represent formalized knowledge about the processes being performed, including context information, definitions, etc. Unstructured data are data produced during execution of those processes. Those data are usually stored in formats like plain comma-separated-values inside TXT or CSV files, without any metadata information. Our solution kept the two types of data physically separated while bridging the gap between them. We stored and managed structured data using OSF, within which knowledge packs are built on top of RDF, in the form of RDF triples. Those triples were queried via SPARQL query language. We stored unstructured data in a time-series database, called InfluxDB, specially designed for this type of data. InfluxDB provided an easy way for querying data through an HTTP endpoint, using a SQL-like query language.

Our main idea was to use the Semantic Web possibility of accessing data through hyperlinks. We embedded hyperlinks to access unstructured databases inside structured RDF models while extending SPARQL query language in order to enable simultaneous processing and exploitation of two types of data. Our extension enabled retrieving time-series data inside newly inserted type of queries called TIMESERIES queries and their afterward processing in enclosing queries.

It is possible to further upgrade our extension in order to make it more flexible in the future. Extending the scope of TIMESERIES queries by making them able to consume response retrieved from a Web service in any kind of standard format (not only JSON) would enable using any kind of data that is available online.

Acknowledgements

The presented work was funded and supported by the Austrian Research Promotion Agency (FFG) (#6112792).

References

Alkhateeb, F., Baget, J., & Euzenat, J. (2007). RDF with regular expressions, Report about research 6191, Institut National de Recherche en Informatique et en Automatique (INRIA). https://hal.inria.fr/inria-00144922v4/document

Almolda, X. (2016). Combining timeseries data with RDF. https://www.melodiesproject.eu/content/how-store-times-series-rdf

Anadiotis, G. (2017). Semantic data lake architecture in healthcare and beyond. https://www.zdnet.com/article/semantic-data-lakes-architecture-in-healthcare-and-beyond/

Angles, R., & Gutierrez, C. (2011). Subqueries in SPARQL, Conference: *Proceedings of the 5th Alberto Mendelzon International Workshop on Foundations of Data Management*, Santiago, Chile. https://pdfs.semanticscholar.org/a323/9db55f3c2e9bc828814c8f2bb4c03f9e8f6b.pdf?_ga=2.4983088.226591044.1553548895-2084923613.1548951967

Barbieri, D.F., Braga, D., Ceri, S., Della Valle, E., & Grossniklaus, M. (2010). C-SPARQL: A Continuous Query Language for RDF Data Streams, *International Journal of Semantic Computing*, *Volume 04*, *No 01*, Pages 3-25, Special Issue: Web Scale Reasoning. https://doi.org/10.1142/S1793351X10000936

Bizer, C., Heath, T., Idehen, K., & Berners-Lee, T. (2008). Linked Data on the Web, *Proceedings of the 17th international conference on World Wide Web*, Pages 1265-1266. https://doi.org/10.1145/1367497.1367760

Bolles, A., Grawunder, M., & Jacobi, J. (2008). Streaming SPARQL - Extending SPARQL to Process Data Streams, *The Semantic Web: Research and Applications. ESWC 2008. Lecture Notes in Computer Science*, *Volume 5021*, Pages 448-462, Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-68234-9_34

Bornea, A.M., Dolby, J., Kementsietsidis, A., Srinivas, K., Dantressangle, P., Udrea, O., & Bhattacharjee, B. (2013). Building an efficient RDF store over a relational database, Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data, Pages 121-132. https://doi.org/10.1145/2463676.2463718

Brickley, D., & Guha, R.V. (2014). Resource Description Framework Schema 1.1 Specification, World Wide Web Consortium Recommendation. https://www.w3.org/TR/rdf-schema/

Cheng, J., Ma, Z.M., & Yan, L. (2010). f-SPARQL: A Flexible Extension of SPARQL, *Database and Expert Systems Applications. DEXA 2010. Lecture Notes in Computer Science*, *Volume 6261*, Pages 487-494, Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-15364-8_41

Cudré-Mauroux, P., Enchev, I., Fundatureanu, S., Groth, P., Haque, A., Harth, A., Keppmann, F.L, Miranker, D., Sequeda, J.F., & Wylot, M. (2013). NoSQL Databases for RDF: An Empirical Evaluation, The Semantic Web – ISWC 2013. ISWC 2013. Lecture Notes in Computer Science, Volume 8219, Pages 310-325, Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-41338-4_20

Hu, S., Corry, E., Curry, E., Turner, W.J.N., & O'Donnell, J. (2016). Building performance optimisation: A hybrid architecture for the integration of contextual information and time-series data, *Automation in Construction*, *Volume 70*, Pages 51-61. https://doi.org/10.1016/j.autcon.2016.05.018

Lefrançois, M., Zimmermann, A., & Bakerally, N. (2017). A SPARQL Extension for Generating RDF from Heterogeneous Formats, The Semantic Web. ESWC 2017. Lecture Notes in Computer Science, Volume 10249, Pages 35-50, Springer, Cham. https://doi.org/10.1007/978-3-319-58068-5_3

Manabu, K., & Yoshiaki, N. (2008). Data-based process monitoring, process control, and quality improvement: Recent developments and applications in steel industry, *Computers & Chemical Engineering*, *Volume 32*, *Issues 1-2*, Pages 12-24. https://doi.org/10.1016/j.compchemeng.2007.07.005

Mayer, S., Hodges, J., Yu, D., Kritzler, M., & Michahelles, F. (2017). An Open Semantic Framework for the Industrial Internet of Things, *IEEE Intelligent Systems*, *Volume 32*, *Issue 1*, Pages 96-101. https://doi.org/10.1109/MIS.2017.9

McBride, B. (2002). Jena: A semantic Web toolkit, *IEEE Internet Computing*, *Volume 6*, *Issue 6*, Pages 55-59. https://doi.org/10.1109/MIC.2002.1067737

Mizell, D., Maschhoff, K.J., & Reinhardt, S.P. (2014). Extending SPARQL with graph functions, *2014 IEEE International Conference on Big Data*, Washington, DC, USA. https://doi.org/10.1109/BigData.2014.7004371

Prud'hommeaux, E., Harris, S., & Seaborne, A. (2013). SPARQL Query Language for Resource Description Framework, World Wide Web Consortium Recommendation. https://www.w3.org/TR/sparql11-query/

Zhiqiang, G., Zhihuan, S., & Furong, G. (2013). Review of Recent Research on Data-Based Process Monitoring, *Industrial & Engineering Chemistry Research*, *Volume 52*, Pages 3543-3562. https://doi.org/10.1021/ie302069q