Technical Report

A Query Language for Multi-version Data Web Archives

Marios Meimaris

m.meimaris@imis.athena-innovation.gr
Institute for the Management of Information Systems
Research Center Athena,
Greece

George Papastefanatos

gpapas@imis.athena-innovation.gr
Institute for the Management of Information Systems
Research Center Athena,
Greece

Stratis Viglas

sviglas@inf.ed.ac.uk
School of Informatics
University of Edinburgh
UK

Yannis Stavrakas

yannis@imis.athena-innovation.gr
Institute for the Management of Information Systems
Research Center Athena,
Greece

Christos Pateritsas

pater@imis.athena-innovation.gr
Institute for the Management of Information Systems
Research Center Athena,
Greece

April, 2015
Institute for the Management of Information Systems,
Research Center Athena,
Greece

Technical Report No: TR-IMIS-2015-2

A Query Language for Multi-version Data Web Archives

Marios Meimaris[1], George Papastefanatos[1], Stratis Viglas[2], Yannis Stavrakas[1], and Christos Pateritsas[1]

[1]Institute for the Management of Information Systems, Research Center "Athena", Greece
{m.meimaris, gpapas, yannis, pater}@imis.athena-innovation.gr
[2]School of Informatics, University of Edinburgh, UK
sviglas@inf.ed.ac.uk

Abstract. The Data Web refers to the vast and rapidly increasing quantity of scientific, corporate, government and crowd-sourced data published in the form of Linked Open Data, which encourages the uniform representation of heterogeneous data items on the web and the creation of links between them. The growing availability of open linked datasets has brought forth significant new challenges regarding their proper preservation and the management of evolving information within them. In this paper, we focus on the evolution and preservation challenges related to publishing and preserving evolving linked data across time. We discuss the main problems regarding their proper modelling and querying and provide a conceptual model and a query language for modelling and retrieving evolving data along with changes affecting them. We present in details the syntax of the query language and demonstrate its functionality over a real-world use case of evolving linked dataset from the biological domain.

Keywords: Data Web, Data Evolution, Linked Data Preservation, Archiving

1 Introduction

The Data Web encompasses the vast and rapidly increasing quantity of scientific, corporate, government and crowd-sourced data being published and interlinked across disparate sites on the web, usually in the form of Linked Open Data (LOD). Data-aware practices, such as data interlinking between heterogeneous sources and data visualization, have a huge potential to create insights and additional value across several sectors, however little attention has been given to the long-term accessibility and usability of open datasets in the Data Web. Linked open datasets are subject to frequent changes in the encoded facts, in their structure, or the data collection process itself. Most changes are performed and managed under no centralized administration, eventually inducing several inconsistencies across interlinked datasets. LOD should be preserved by keeping them constantly accessible and integrated into a well-designed framework for evolving datasets that offers functionality for versioning, provenance tracking, change detection and quality control while at the same time provides efficient ways for querying the data both statically and across time.

Most of the challenges related to the management of LOD evolution stem from the decentralized nature of the publication, curation and evolution of interdependent datasets, with rich semantics and structural constraints, across multiple disparate sites. Traditional database versioning imposes that data and evolution management take place within well-defined environments where change operations and data dependencies can be monitored and handled. On the other hand, web and digital preservation techniques assume that preservation subjects, such as web pages, are plain digital assets that are collected (usually via a crawling mechanism), time stamped and archived for future reference. In contrast to these two approaches, the Data Web poses new requirements for the management of evolution [11][24]. Observe Figure 1 where an example from the biological domain is presented. EFO is an ontology that combines parts of several life science ontologies, including anatomy, disease and chemical compounds [25]. Its purpose is to enable annotation, analysis and visualization of data related to experiments of the European Bioinformatics Institute[1]. In the figure, a URI that represents a Cell Line class changes between two consecutive versions and becomes obsolete. EFO entities are being published in LOD format, enabling other sites to reference and interlink with them. EFO is regularly updated and new versions are published on the web, usually overwriting previous ones. In this context, several interesting problems and challenges arise related to long-term preservation and accessibility of evolving LOD datasets:

Evolution Modeling: LOD datasets are evolving entities for which additional constraints may hold related to the way data is published, and evolve as dictated by domain-specific, complex changes. *This calls for appropriate modelling methods for preserving across time a multitude of dimensions related to the internal structure of a dataset, its content and semantics as well as the context of its publication*. Preservation should exhibit format-independence, data traceability and reproducibility and a common representation for data that originate from different models. Reference schemes (URIs) must be properly assigned such that unique identification and resolution is achieved across different sites, and most importantly across time. Provenance metadata can capture dataset lineage from the dataset to the record level. Distributed replication of LOD enhanced with temporal and provenance annotations can enable long-term availability and trust.

Change management. Changes can occur at different granularity levels. At the dataset level, datasets are added, republished, or even removed, without versioning or preservation control; at the schema level, the structure may change calling for repair and validation on new versions; finally, at the instance level data resources and facts are added, deleted or updated. Discovering changes [16] and representing them as first class citizens with structural, semantic, temporal and provenance information *is vital in various tasks such as the synchronization of autonomously developed LOD versions, or visualizing the evolution history of a particular dataset*.

[1] http://www.ebi.ac.uk/

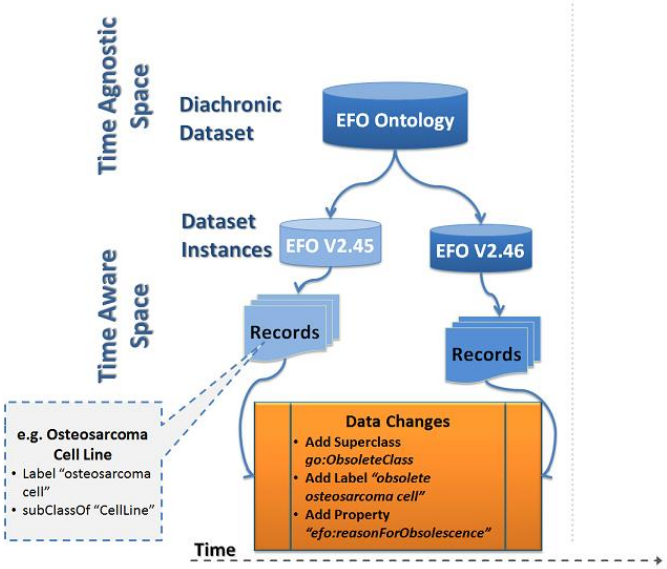

Figure 1. Evolution of a Cell Line between versions 2.45 and 2.46 of the Experimental Factor Ontology.

Longitudinal accessibility and querying. LOD preservation mechanisms must enable the long-term accessibility of datasets and their meaningful exploration over time. Datasets with different time and schema constraints coexist and must be uniformly accessed, retrieved and combined. Longitudinal query capabilities must be offered such that data consumers can answer several types of queries, within a version or across sets of versions. Querying must take place (i) across time, (ii) across datasets and (iii) across different levels of granularity of evolving things.

Considering the above, the benefits of evolution management can be placed into two categories, namely quality control and data analysis. Data evolution provides valuable insights on the dynamics of the data, their domains and the operational aspects of the communities they are found in, while tracking the history of and maintaining proper metadata of data objects across time enables better interoperability, trust and data quality.

To address these challenges, in this paper we propose a modelling approach and a query language for evolving LOD datasets. At the basis of the archive lies a conceptual model, called DIACHRON model[2] that captures structural concepts like datasets and their schemas, semantics like web resources, their properties and links between them as well as changes occurring on these concepts in different granularity levels. In the same time, our approach models in a uniform way both time-aware (evolving) and time-agnostic (diachronic) concepts, representing their between interconnections. Based on this model, a query language is designed that specifically caters for the

[2] It has been developed in the context of the DIACHRON project and is part of the DIACHRON preservation platform, http://www.diachron-fp7.eu

model's inherent characteristics and takes advantage of the abstraction levels thus making the user avoid complicated, implementation-dependent queries. The query language is designed as an extension of SPARQL, specific to the DIACHRON model, that tackles the duality of data (evolving vs. diachronic objects) in order to provide a query mechanism with the ability to correlate source data with changes, annotations at various levels and other kinds of DIACHRON related metadata across time. Finally, these are implemented under the broader scope of an archiving framework capable of storing and making available in the long term evolving LOD datasets.

Our approach provides the following contributions:

1. *Evolution Storage*: We consider the way data is recorded at the source as an evolving aspect of a dataset that enables recreation of a dataset at its original model and format. Furthermore, storing resources and their semantics and capturing their evolution enables semantically meaningful tracking of the resources' timelines. We show how these aspects can be modelled stored to support querying.
2. *Multi-versioning*: We propose a way to version datasets on different levels by providing both time-agnostic and time-dependent representations of evolving entities.
3. *Metadata management*: We provide placeholders for metadata at all levels through the DIACHRON model, which enable provenance and temporal annotations on all types of objects within the archive, such as the structural blocks of a dataset, the higher-level entities that appear in it, as well as the changes between versions.
4. *Querying*: We propose the DIACHRON Query Language in order to enable retrieval of data and metadata with complex queries across versions and integration of low level and high level data as well as changes in the results.

This paper is outlined as follows. In section 2 we present the DIACHRON data model, in section 3 we present the DIACHRON query language, in section 4 we describe our implementation of an archive that uses the proposed model and query language, while in section 5 we perform experimental evaluation. Section 6 discusses related work and section 7 concludes the paper.

2 An archive model for evolving datasets

Our modelling approach supports a format-independent archiving mechanism that maintains syntactic integrity by making sure that the original datasets are reproducible and at the same time takes advantage of information-rich content in these datasets. Format-independence of the model is part of a larger scope of requirements within DIACHRON where different source models (e.g. relational, multidimensional, ontological) can be transformed to the same RDF representation, uniformly annotated with temporal and provenance information and archived.. The DIACHRON model provides the basis for defining semantically richer entities that evolve with respect to their source datasets' history. At the core of the model lies the notion of the *evolving entity*, which captures both structural and semantic constructs of a dataset and acts as a common placeholder for provenance, temporal, and other types of metadata.

Evolving entities are identifiable and citable objects. These entities all share a common ancestor (i.e. are subclasses of) the Diachronic Entity, which allows the aforementioned requirements to be addressed on many different levels. The different types of entities in the DIACHRON model and their interactions can be seen in Fig 2 and described in the following.

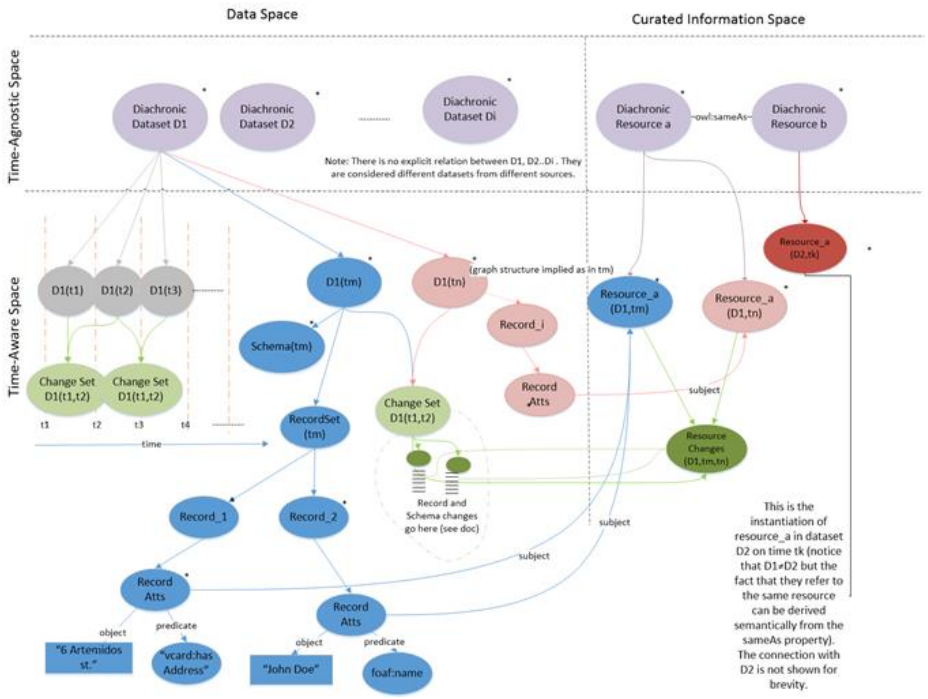

Figure 2. The DIACHRON model space.

Diachronic datasets and dataset instantiations. Diachronic datasets are conceptual entities that represent a particular dataset from a time-agnostic point of view, which in turn is linked to its temporal instantiations or versions. Furthermore, diachronic dataset metadata comprise information that is not subject to change, such as diachronic dataset identifiers. These identifiers serve as ways to refer to the datasets in a time and/or version unaware fashion (i.e. diachronic citations). On the other hand, dataset instantiations define temporal versions of diachronic datasets, holding information on how and when a particular dataset was relevant, active, trusted and so on..

Record sets. Record sets are collections of data entries (e.g. tuples, triples) over a given subject/primary key within a particular dataset instantiation. Given a record set and the dataset's metadata information, the dataset instantiation can be queried and reproduced in its original form. Keeping data objects separate from schema objects makes versions interpretable by different schemata (e.g. new schema on old data or vice versa).

Schema Objects. Schema objects represent the schema-related entities of the archived datasets given the dataset's source model. For instance, the classes along with

their class restrictions of an ontology, the properties and their definitions (domains, ranges, meta properties depending on the expressivity) are modelled as schema objects. Similarly to data objects, the goal is to provide a reusable modelling mechanism for identifying and referring to schema elements and their evolution across datasets. In this way, schema evolution is captured by annotating schema elements with schema changes.

Data Objects. Data objects consist of *records* and *record attributes*. A record represents a most granular data entry about a particular evolving entity. Records are uniquely identified in order to make record-level annotation feasible in order to attribute provenance, temporality and changes on them. A record serves as a container of one or more record attributes. Every data record is broken down to assertions (facts) that can be expressed as RDF triples. In this sense, a record reifies the predicate-object pairs for a fixed subject. These predicate-object pairs are called record attributes. For instance, a tuple from a relational table is considered to be a record describing the tuple's primary key, with each relational attribute being a record attribute. In [3], [26] we describe in details how data records from relational, multidimensional and RDF models can be mapped to data objects in our model.

Diachronic Resources and resource instantiations. Similarly to diachronic datasets, a diachronic resource represents a time-agnostic information entity. The resource instantiation captures the resource evolution across time and its realization over a versioned dataset's records. The definition of a resource consists of two parts; the resource identification definition gives the way an instantiated resource is identified within the archive. The resource description definition provides the way a resource is evaluated over the records of a particular dataset instantiation. Resources can be versatile in nature across datasets and data formats. For example, given an ontology and its instantiation, each class instance can describe a resource identified by the respective URI. Given a table of employees in a relational database, a resource in this sense can be a particular employee identified by his primary key. Finally, in a multidimensional dataset, a resource can be a specific observation identified by the values of the constituent dimensions. More complex definitions of resources are allowed and, in fact, encouraged for capturing more high-level, curator specific semantics of evolution and dataset dynamics.

Change sets. Changes come in Change Sets between two dataset instantiations of a diachronic dataset. These are comprised of changes between record sets, changes between schemata and changes between resource instantiations of the two datasets under comparison.

The proposed data model provides a conceptual way of uniformly representing low-level and high-level evolving entities. Within the context of our model, an evolving entity is a dataset instantiation (affected by changes in its schema and contents), a schema object, a data object or finally a resource instantiation object. This gives us a uniform way to model evolution and annotate entities at different levels of granularities with information related to the changes affecting them. Furthermore, it enables us to enrich evolving entities with metadata related to the way these entities are published on remote sites and collected in the archive, such as provenance information, quality and trust.

3 The DIACHRON Query Language

3.1 Requirements and Overview

As described is the previous section, the DIACHRON model provides metadata hooks in many different granularities, from the dataset to the record level. In this section, we motivate the need for an appropriate query language that exploits the specificities of the data model and provides ways to achieve the following:

- *Dataset and version listing*: Retrieve lists of datasets stored in the archive, as well as lists of the available versions of a given dataset. These can either be exhaustive or filtered based on temporal, provenance or other metadata criteria.
- *Data queries*: Retrieve part(s) of a dataset that match certain criteria.
- *Longitudinal queries*: As above but with the timeline of all types of diachronic entities. Temporal criteria can be applied to limit the timeline (specific versions or time periods), or successive versions.
- *Queries on Changes*: Retrieve changes between two concurrent versions of an entity (dataset, resource etc.). Limit results for specific type of changes, or for a specific part of the data.
- *Mixed Queries on Changes and Data*: Retrieve datasets or parts of datasets that are affected by specific types of changes.

Furthermore, in this section, we propose the DIACHRON Query Language (DIACHRON QL), an extension of SPARQL, to tackle these requirements. On the basis of the query language is the DIACHRON graph pattern, a specialization of a SPARQL graph pattern. SPARQL queries are valid DIACHRON queries, however several new keywords are defined in order to cover the model's characteristics and allow the user to query archived data intuitively and seamlessly. The DIACHRON query language introduces keywords that can define the scope of a query w.r.t. the matched diachronic datasets and its versions, or the change sets used to match changes. This is done with the use of `FROM DATASET` or `FROM CHANGES` in the beginning of a `SELECT` or `CONSTRUCT` query, which are both used to define the URI(s) of the dataset versions or change sets where the query body should be limited. While `FROM DATASET` and `FROM CHANGES` are used outside of the query body in order to define the scope for the following query, limiting the scope of a specific graph pattern can be done inside the query body, similarly to SPARQL's GRAPH. In DIACHRON, this is done by using `DATASET` or `CHANGES` followed by a graph pattern in curly brackets (notice that `FROM` is used only outside of the query body). Specific dataset/change set versions and version intervals can be defined using the `AT VERSION`, `AFTER VERSION`, BEFORE `VERSION` or `BETWEEN VERSIONS` keywords.

Diachronic datasets, versions and change sets can be bound to variables at query execution with the use of `DATASET` or `CHANGES`. This is simply done by using variables instead of explicit URIs, inside the query body, i.e. not in a `FROM` clause. For example, consider the case where we want to retrieve all the information (predicate-

object pairs) associated with the protein *efo:EFO_0004626*, and find out what the state of this information is for all the dataset versions of the EFO ontology it appears in (and what are those versions). That is, the dataset versions as well as the actual information are to be retrieved. In DIACHRON QL this can be written as follows:

```
SELECT ?version ?p ?o WHERE {
     DATASET <EFO> AT VERSION ?version {
            efo:EFO_0004626  ?p  ?o
     }
}
```

This will retrieve all versions of EFO joined with predicate-object pairs for the protein *efo:EFO_0004626*. If we want to retrieve the records these predicate-object pairs appear in, without querying for the particular dataset versions. We can retrieve the URIs of the DIACHRON records these triples appear in by modifying the query as follows:

```
SELECT ?rec ?p ?o FROM DATASET <EFO> WHERE {
            RECORD ?rec {efo:EFO_0004626  ?p  ?o}
}
```

With the optional use of the `RECATT` keyword we can retrieve the URIs of the record attributes of a matched record. The previous query would become:

```
SELECT ?rec ?ra ?p ?o FROM DATASET <EFO> WHERE {
            RECORD ?rec {efo:EFO_0004626
                  RECATT ?ra {?p  ?o}
            }
}
```

When writing a basic archive graph pattern, the query can either contain simple triples, or more verbose constructs that take into account the archive data model and structure. Specifically, the simple triples will match the de-reified data, whereas the `RECORD` and `RECATT` (abbreviation of '*record attribute*') blocks will also take into account a triple's record or record attribute.

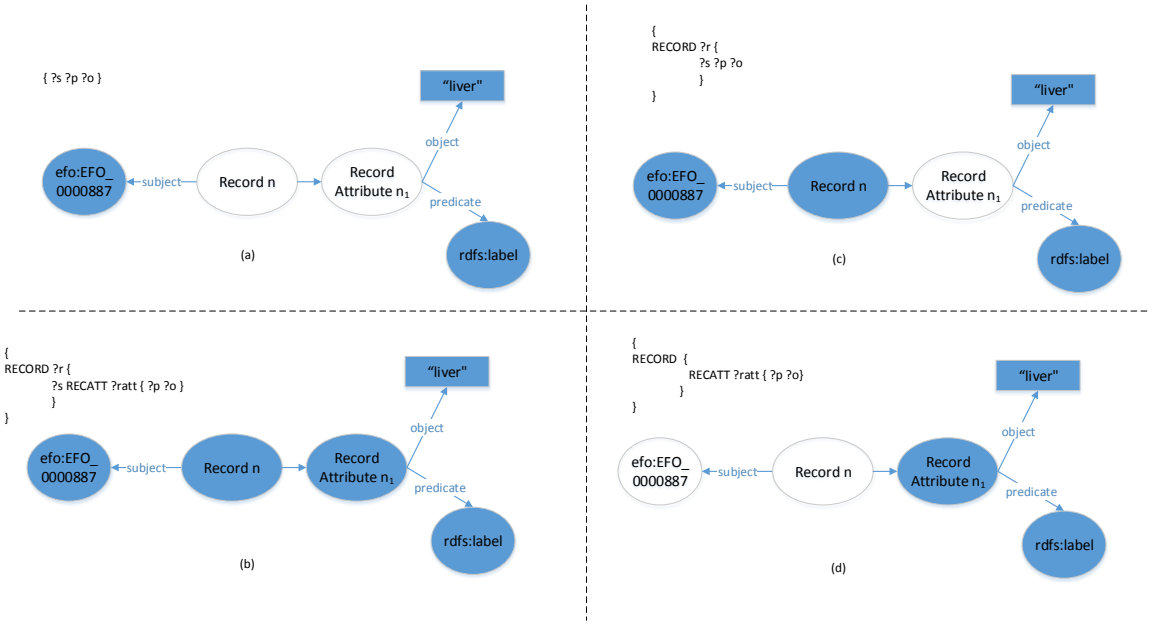

Figure 3. (a) matches in a simple triple query, (b) matches a blown-out version of the same query with the RECORD and RECATT terms, selecting both data and structural elements. (c) matches subject, predicate, object and record, (d) matches predicate, object and record attribute.

This is detailed in figure 3 where we show how term and variable use is reflected on the matched graph of a particular reified triple. This way, metadata (e.g. temporal, provenance) of the records and/or record attributes can be queried as well as combined with data queries. It should be noted that in the simplest case where only the data are of interest, the query does not need to include `RECORD` or `RECATT` blocks.

3.2 Query Syntax and Examples

DIACHRON QL clauses are formally described in the following section and an overview of them is presented in table 1. In table 2 usage examples are presented for all DIACHRON QL clauses.

`FROM DATASET <diachronicURI> [[AT VERSION <instantiationURI>]]`

The `FROM DATASET` keyword is followed by a URI of a diachronic dataset to declare the dataset scope of the query. If no `FROM DATASET` is given, then the whole corpus of datasets is queried. The optional `AT VERSION` keyword limits the selected diachronic dataset to a specific dataset instantiation. No variables can be given in any of the parameters of `FROM DATASET AT VERSION`.

Table 1: The DIACHRON query language syntax in E-BNF.

DiachronQuery :=	‘DIACHRON‘ ‘SELECT’ (‘DISTINCT‘)? (Var+|’*’) *Source_Clause** ‘WHERE‘ *Where_Clause**
Source_Clause :=	(‘FROM DATASET’ <*URI*> [‘AT VERSION’ <*URI*>] | ‘FROM CHANGES’ <*URI*> [‘BEFORE VERSION’ <*URI*> | ‘AFTER VERSION’ <*URI*> | ‘BETWEEN VERSIONS’ <*URI*>+2])
Where_Clause :=	(*Diachron_Pattern* *[‘UNION’ Diachron_Pattern]* *[‘OPTIONAL‘ Diachron_Pattern])*
Diachron_Pattern :=	(*Source_Pattern Basic_Archive_Graph_Pattern*)
Source_Pattern :=	((‘DATASET‘ <VarOrURI> [‘AT VERSION’ <VarOrURI>]) | (‘CHANGES’ <VarOrURI> [‘BEFORE VERSION’ <VarOrURI>]) | (‘CHANGES’ <VarOrURI> [‘AFTER VERSION’ <VarOrURI>]) | (‘CHANGES’ <VarOrURI> [‘BETWEEN VERSIONS’ <VarOrURI>+2]))
Basic_Archive_ Graph_Pattern :=	‘{‘ *SPARQL_Triples_Block** *Record_Block** *Change_Block** ‘}’
Record_Block :=	‘RECORD‘ <VarOrURI> ‘{‘ <VarOrURI> ((<VarOrURI>+2 ‘.’)*) | (‘RECATT‘ <VarOrURI> ‘{‘ <VarOrURI>+2 ‘}’)* ‘}’
Change_Block :=	‘CHANGE‘ <VarOrURI> ‘{‘ (<VarOrURI>+2 ‘.’)* ‘}’
SPARQL_ Triples_Block :=	As defined in the SPARQL recommendation[3].

FROM CHANGES <diachronicURI> [[BETWEEN VERSIONS <version1URI> <version2URI>] || [BEFORE VERSION <versionURI>] || [AFTER VERSION <versionURI>]]

FROM CHANGES is used to query change-sets directly. Optionally, it is immediately followed by a URI of a diachronic dataset that defines the diachronic dataset to be queried on its changes. If no URI is given, then all existing change sets will be used to

[3] http://www.w3.org/TR/sparql11-query/

match the query body. `FROM CHANGES` can optionally be used with `BETWEEN VERSIONS`, `BEFORE` or `AFTER VERSION` to limit the scope of the changes.

`DATASET <URI | ?var> [[AT VERSION <URI | ?var>]] { (query) }`

The `DATASET` keyword differs from `FROM DATASET` in that it is found inside a query body. It is followed by a URI/variable of a diachronic dataset to declare or bind the scope of the graph. `DATASET` is inside a `WHERE` statement and is followed by a graph pattern, on which the dataset restriction is applied. It is optional, meaning that if no `DATASET` is given, then the whole corpus of datasets will be queried, or the datasets defined in the `FROM DATASET` clause. The `AT VERSION` keyword, when applied to a `DATASET` statement inside a `WHERE` clause, is used to either define a specific dataset instantiation or bind dataset instantiations to a variable for the graph pattern that follows. However, `AT VERSION` is optional and if no specific dataset instantiation URI or variable is declared, `AT VERSION` is omitted. An example of matching both triples and versions can be seen in fig 4.

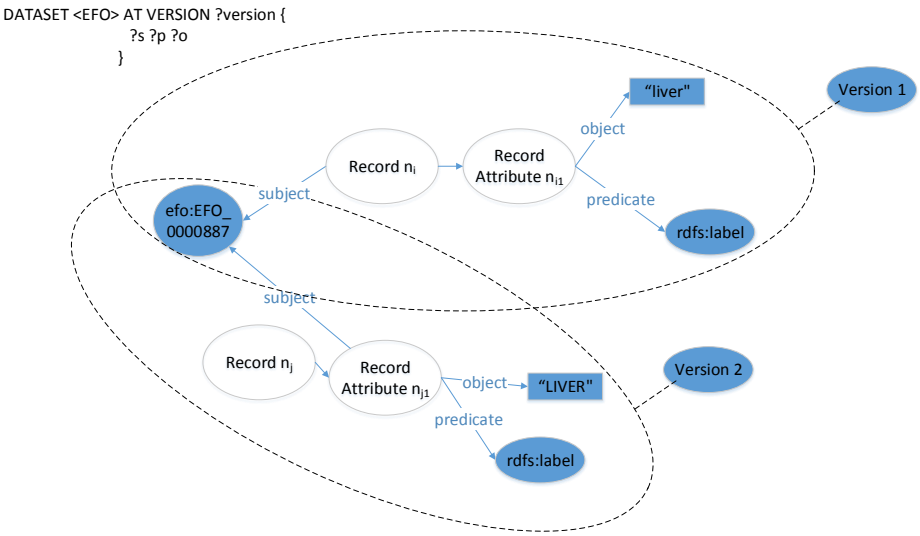

Figure 4. Matching a reified triple in a query with variable versions. Blue nodes are selected by the query.

`RECORD <record_URI | ?record_var>`
`{<subjectURI | ?subject_var > ATTRIBUTE_pattern}`

`RECORD` is used inside the body of a graph pattern for querying either a specific DIACHRON record or to match DIACHRON records in the pattern. It is followed by a record URI/variable. If neither of those is declared, the `RECORD` keyword can be omitted. Following `RECORD` is a block containing a graph pattern that can either be of SPARQL form, or used in conjunction with the `RECATT` keyword.

Table 2. Query language keywords and usage examples.

Keyword	Parameters	Usage example
SELECT	variable list	SELECT ?x, ?y, ?z
FROM DATASET	URI of diachronic dataset	SELECT ?x, ?y, ?z FROM DATASET <efo-protein-sample>
FROM DATASET AT VERSION	URI of dataset instantiation	SELECT ?x, ?y, ?z FROM DATASET <efo-protein-sample> AT VERSION <v1>
FROM CHANGES	URI of diachronic dataset	SELECT ?x, ?y, ?z FROM CHANGES <efo-protein-sample>
FROM CHANGES ... BETWEEN VERSIONS (params)	URIs of dataset instantiations to define the change scope	SELECT ?x, ?y, ?z FROM CHANGES <efo-protein-sample> BETWEEN VERSIONS <v_m>, <v_n>
FROM CHANGES … **AFTER / BEFORE VERSION** (params)	URI of dataset instantiation to define the start/end of the change scope	SELECT ?x, ?y, ?z FROM CHANGES <efo-protein-sample> AFTER / BEFORE VERSION <v_m>

WHERE { (params) }	DIACHRON patterns	SELECT ?x, ?y, ?z
		FROM DATASET <efo-protein-sample>
		WHERE {
		?x a efo:Protein ; ?y ?z .
		}

DATASET (params)	URI or variable of diachronic dataset	SELECT ?x, ?y
		WHERE {
		DATASET ?x {
		?s a efo:Protein.
		}
		DATASET ?y {
		?s dcterms:creator “EBI”
		}
		}

DATASET … **AT VERSION** (params)	URI or variable of dataset instantiation	SELECT ?x, ?y
		WHERE {
		DATASET ?x AT VERSION ?var {
		?s a efo:Protein.
		}
		DATASET ?y AT VERSION <v1> {
		?s dcterms:creator “EBI”
		}
		}

RECORD (params)	URI or variable of DIACHRON record	SELECT ?x, ?r, ?y
		WHERE {
		DATASET ?x AT VERSION ?var {
		RECORD ?r {?s a efo:Protein}
		}
		DATASET ?y AT VERSION <v1> {
		?s dcterms:creator “EBI”
		}
		}

RECATT (params)	URI or variable of a DIACHRON record attribute	SELECT ?var, ?r, ?ra
		WHERE {
		DATASET <efo> AT VERSION ?var {
		RECORD ?r {
		?s RECATT ?ra {rdf:type efo:Protein}
		}
		}
		}

CHANGES (params)	URI of diachronic dataset or variable	SELECT ?c, ?param1, ?value1
		WHERE {
		CHANGE ?c {?param1 ?value1 }
		}

CHANGES ... **BETWEEN VERSIONS** (params)	URIs of dataset instantiations or variables to define the change scope	SELECT ?v1, ?v2, ?c
		WHERE {
		CHANGES <EFO> BETWEEN VERSIONS ?v1,
		?v2 {
		?c rdf:type co:Add_Definition ;
		?p1 [co:param_value ?o3 . rdf:type co:ad_n1] ;
		?p2 [co:param_value ?o4 . rdf:type co:ad_n2]
		}
		}

CHANGES … **AFTER / BEFORE VERSION** (params)	URI of dataset instantiation or variable to define the start/end of the change scope	SELECT ?s ?p ?o
		WHERE {
		CHANGES <efo-protein-sample> BEFORE/AFTER VERSION <v_m> { ?s ?p ?o}
		}
		}

CHANGE (params)	URI of change or variable	SELECT ?v1, ?v2, ?c, ?p ?o WHERE {
		CHANGES <EFO> BETWEEN VERSIONS ?v1
		?v2{
		CHANGE ?c {?p ?o}
		}

RECATT <recattURI | ?recatt_var>
{ <predicateURI | ?predicate_var> <objectURI | ?var> }

RECATT (short for RECORD ATTRIBUTE) is used inside a RECORD block and separates the subject of a DIACHRON record with the record attributes that describe it. It is followed by a URI/variable. If no specific record attribute needs to be queried or matched in a variable, RECATT can be omitted.

CHANGES <diachronicURI | var> [[BETWEEN VERSIONS <version1URI | ?var1>] || [BEFORE VERSION <versionURI | var1>] || [> AFTER VERSION <versionURI | var1>]]

CHANGES is used to limit the scope of a block within a larger query into a particular change set, or match change sets to a variable. If no URI is given, then all existing change sets will be used to match the query body. CHANGES can optionally be used with BETWEEN VERSIONS, BEFORE VERSION or AFTER VERSION to limit the scope of the changes or bind the dataset versions that match the change set pattern to variables.

CHANGE <changeURI | ?change_var>

The CHANGE keyword is used to query a particular change in a fixed query block within a larger query pattern. It is followed by a specific change URI or a variable to be bound. The succeeding block is used to declare the change parameters in a predicate-object manner.

4 Implementation

In this section we present the implementation of the proposed query language. We first provide an overview of the overall architecture of the DIACHRON archive. The archive employs the proposed DIACHRON model for storing evolving LOD datasets. The query engine is a core component of the archive, responsible for processing queries expressed in the DIACHRON QL and retrieving data out of the archive.

4.1 Overall architecture

The architecture of the archive and various components of the archive can be seen in figure 5. The archive's web service interface is exposed via the HTTP protocol as the primary access mechanism of the archive through a RESTful web service API. The Data Access Manager provides low level data management functionality for the archive. It is bound to the specific technology of the underlying store, in our case Openlink Virtuoso 7.1[4], as well as external libraries that provide data access functionality for third-party vendors. For this we used the Jena semantic web framework[5]. It serves as an abstraction layer between the store and the query processor.

[4] http://virtuoso.openlinksw.com/
[5] https://jena.apache.org/

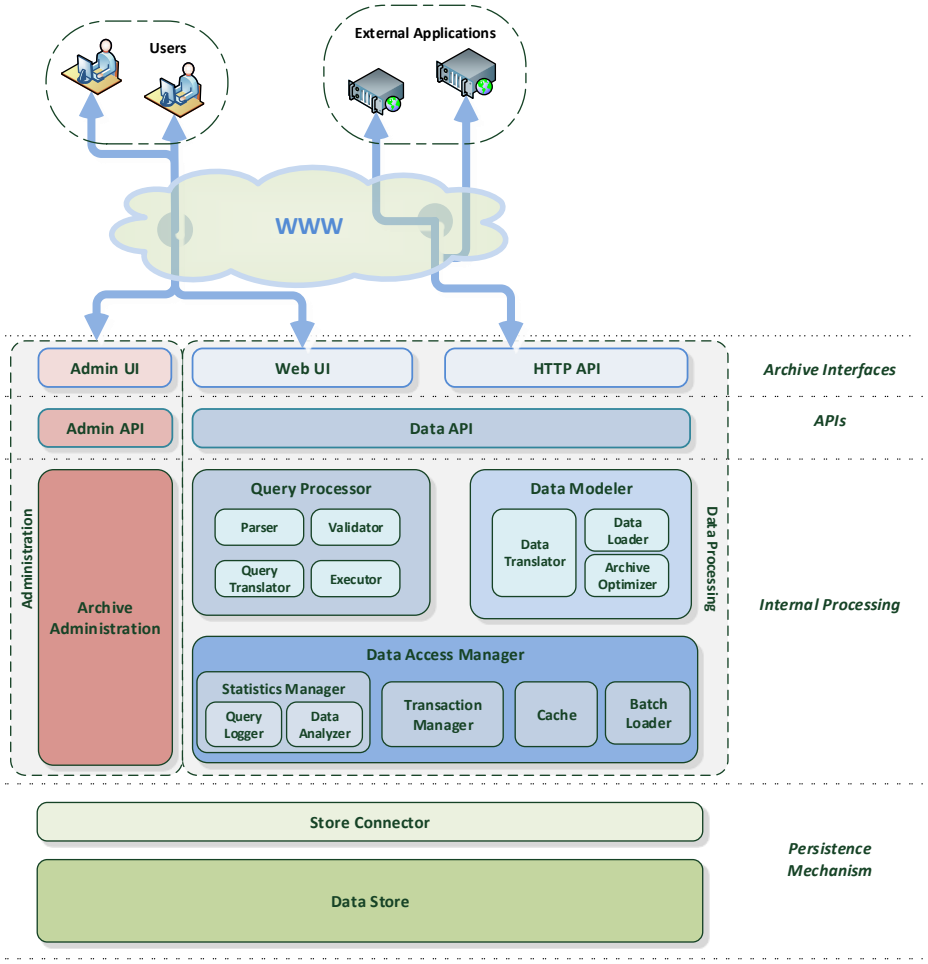

Figure 5: Architecture of the archive.

The archive employs a Data Access Manager, a Store Connector, a Data Modeler, an Archive Optimizer and a Query Processor. The Store Connector is the software package that provides an API to other components of the archiving module for communication and data exchange with the underlying store and is implemented with the Virtuoso JDBC Driver package[6]. The Data Store employs a Virtuoso 7.1 instance. The Data Modeler component handles the dataset input functionality and data transformations from the DIACHRON dataset model to the native data model of the store and vice versa, and consists of the Data Translator and the Data Loader. The Archive optimizer component supports the optimization of the datasets' storage method based on various archive strategies as shown in [23] that are not discussed in this paper. It

[6] http://docs.openlinksw.com/virtuoso/VirtuosoDriverJDBC.html

performs analysis of the dataset characteristics and chooses the most efficient storage strategy based on metrics.

The Query Processor component is the base mechanism for query processing and thus data access. It consists of the following subcomponents:

- **Validator:** validates the DIACHRON queries for syntactic validity against the DIACHRON QL syntax described in section 3.
- **Query parser:** parses the queries in DIACHRON QL so as to create a structure of elements that correspond to DIACHRON Dataset Entities and DIACHRON query operators.
- **Query Translator:** creates the execution plan of DIACHRON queries by translating the queries in SPARQL. The translator also makes use of the various archive structures implemented in the persistence store and the appropriate indexes and dictionaries. The query translator is the subcomponent that ties the DIACHRON archive module to the specific storage technology of RDF and SPARQL. Translation is further described in section 4.2
- **Executor:** executes the created execution plan step by step and retrieves the raw data from the store so as to build the result set of the query. It uses also the Data Modeler component in order to perform, if necessary, data transformations from the native data model of the underlying store to the DIACHRON dataset model.

4.2 Translation of DIACHRON QL to SPARQL

DIACHRON graph patterns are translated to SPARQL as shown in table 3. DIACHRON QL can be directly translated to SPARQL in order to take advantage of existing foundations, concepts and implementations and benefit from the W3C recommended graph-based query language. The language is tightly bound to the data model of evolution presented in this article and the new syntax tokens correspond to SPARQL query patterns that are populated with appropriate variable bindings and/or URIs. The drawback to this approach lies in the fact that to actually translate a given query language to SPARQL, a good grip of the archive's internal structure is needed. In particular, in our implementation, datasets, record sets and change sets are all stored in designated named graphs. Therefore, to actually translate a query that defines a particular dataset version (e.g. by using the DATASET AT VERSION token), the knowledge that this version's record set and schema set are stored in named graphs is needed in order to provide it to the resulting SPARQL query's FROM or GRAPH pattern. If the archive was implemented on top of a relational database, the query language translator would need to know the database's schema, the names and characteristics of the tables where records and other evolving entities are stored. Therefore, this makes it reliant to the underlying implementation.

Table 3. DIACHRON graph patterns and their translation to SPARQL.

DIACHRON Pattern (Parsed Syntax)	SPARQL
{?s ?p ?o}	{ [a evo:Record ; evo:subject ?s ; evo:hasRecordAttribute [evo:predicate ?p ; evo:object ?o]]}

{
RECORD ?r {?s ?p ?o}
}
	{?r a evo:Record ;
	evo:subject ?s ;
	evo:hasRecordAttribute
	[evo:predicate ?p ; evo:object ?o]}

{ RECORD ?r {
?s RECATT ?ra {?p ?o}
}
}
	{?r a evo:Record ;
	evo:subject ?s ;
	evo:hasRecordAttribute ?ra .
	?ra evo:predicate ?p ;
	evo:object ?o}

{
DATASET <EFO> AT VERSION ?v
{
RECORD ?r {
?s RECATT ?ra {?p ?o}
}
}
}
	{GRAPH <dataset_dictionary> {
	<EFO> evo:hasInstantiation ?v .
	?v evo:hasRecordSet ?rs
	} GRAPH ?rs{
	?r a evo:Record ;
	evo:subject ?s ;
	evo:hasRecordAttribute ?ra .
	?ra evo:predicate ?p ;
	evo:object ?o }}

FROM DATASET <EFO> AT VERSION <EFO/v1>
 	
{
RECORD ?r {
?s RECATT ?ra {?p ?o}
}
}
	{GRAPH <dataset_dictionary> {
	<EFO> evo:hasInstantiation <EFO/v1> .
	<EFO/v1> evo:hasRecordSet ?rs
	} GRAPH ?rs{
	?r a evo:Record ; evo:subject ?s ;
	evo:hasRecordAttribute ?ra .
	?ra evo:predicate ?p ;
	evo:object ?o }}

FROM CHANGES <EFO> BETWEEN VERSIONS <EFO/v1> <EFO/v2>
 	
{
CHANGE ?c {?p ?o}
}
	{GRAPH <dataset_dictionary> {
	?cs a evo:ChangeSet ;
	evo:oldVersion <EFO/v1> ;
	evo:newVersion <EFO/v2>
	} GRAPH ?cs{
	?c a _:Change ; ?p ?o }}

5 Experimental Evaluation

In this section we present the experimental evaluation of our approach over a real world evolving biological use case of the EFO ontology as well as use case concerning evolving multidimensional data of the statistical domain published on the web in LOD format following the Data Cube Vocabulary[7] approach. In the first case, we

[7] http://www.w3.org/TR/vocab-data-cube/

consider 15 consecutive versions of the ontology, that exhibit various types of changes, both simple and complex. In the second case, we consider four multidimensional datasets each comprised of three consecutive versions. We load all datasets into the same archive instance, and in order to do so, the data are first converted to fit the DIACHRON model. For this, we implemented a conversion mechanism as part of the Data Modeller component presented in the previous section. The modeller reifies data to records and record attributes. Data are mapped to the DIACHRON data model in the following manner. First, classes and their definitions (domains, ranges) are modelled as schema objects. The triples are grouped by their subjects. For each subject URI, its corresponding predicate-object pairs are modelled as record attributes and grouped in records. The subject records are in turn connected with the record attributes created for each triple associated with a subject URI. For a more in-depth discussion of the mapping process the reader is referred to [26].

5.2 Experiments

The goal of the experimental evaluation was to assess the performance of our implementation w.r.t three main aspects: the time overhead related to the initial loading of the archive, the time overhead related to the retrieval of the datasets in their original form (de-reification and serialization) and the time overhead of executing queries of different difficulty. Our approach was implemented in Java 1.7, and all experiments were performed on a server with Intel i7 3820 3.6GHz, running Debian with kernel version 3.2.0 and allocated memory of 8GB.

First, bulk operations on whole datasets have been tested, namely loading and retrieving full dataset versions. Loading and retrieval times can be seen in figure 6 (a) and (b). A series of 10 tests were run for each version of the datasets and the averages have been used in computing times, using least squared sums. Loading a dataset in the archive implies splitting it into the corresponding structures, i.e. dataset, record set, schema set and change set, and storing it in different named graphs. The splits were done directly in the store using the SPARQL update language and basic pattern matching, thus no need to put a whole dataset in memory arose, which would be costly in terms of loading in and building the respective Java objects in Jena[8]. The increasing sizes of the input datasets are the effect of their evolution, as new triples are being added. In the same figure (b), retrieval times can be seen for the same datasets. Retrieval of a dataset is the process of de-reifying it to recreate the dataset version at its original form and structure. As can be seen, both loading times and retrieval fit into a linear regression w.r.t to the datasets' sizes as measured in record attributes and imply that no additional time overhead is imposed that would destroy linearity as new versions of a dataset are stored in the archive.

Figures 6(c)-(f) show running times of 12 queries we devised for this experiment. An analysis of the queries' characteristics can be seen in table 4. In figures 6 (c) and (d) we perform a series of queries on different dataset versions. Specifically, two sets of 5 queries have been devised to run on a fixed dataset. Each query is run on one

[8] https://jena.apache.org/

particular version, and the total running time of all 5 queries in each set (c) and (d) is calculated after retrieving the results and storing them in memory, which implies a simple iteration on all results. The query sets are made up from SELECT queries that combine structural entities (records, record attributes etc.) with actual data entries (subject URIs etc.) in different levels of complexity. In Figure 6 (a) no aggregate functions, OPTIONALs or other complex querying capabilities have been used, while in Figure 6 (b) the queries consist of selecting, aggregating and filtering graph patterns. As in the case of loading and retrieval, the archive behaves in a linear way as the size of a dataset increases.

Finally, two queries, Q11 and Q12, with variable datasets that search in the entire archive have been devised and run on an incrementally larger archive, that is, the queries have been tested on deployments of the archive where versions of datasets are being incrementally added to their corresponding diachronic datasets. The queries use dataset versions as variables. The results can be seen in Figure 6 (e) and (f) where linearity is still being preserved when new datasets are stored.

Table 4. Characteristics of the experiment queries.

	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Q11	Q12
DISTINCT	√	√	√	√	√	√	√	√	√	√	√	√
Unbound predicates								√	√	√	√	√
Filters											√	√
Aggregate Functions							√	√	√	√	√	√
ORDER BY							√	√	√	√	√	√
OPTIONAL												√
SELECT	√	√	√	√	√	√	√	√	√	√		√
CONSTRUCT											√	
Reified data	√	√	√	√	√							
De-reified pattern						√	√	√	√	√	√	√
Diachronic metadata					√	√			√	√	√	√
Unbound named graphs											√	√

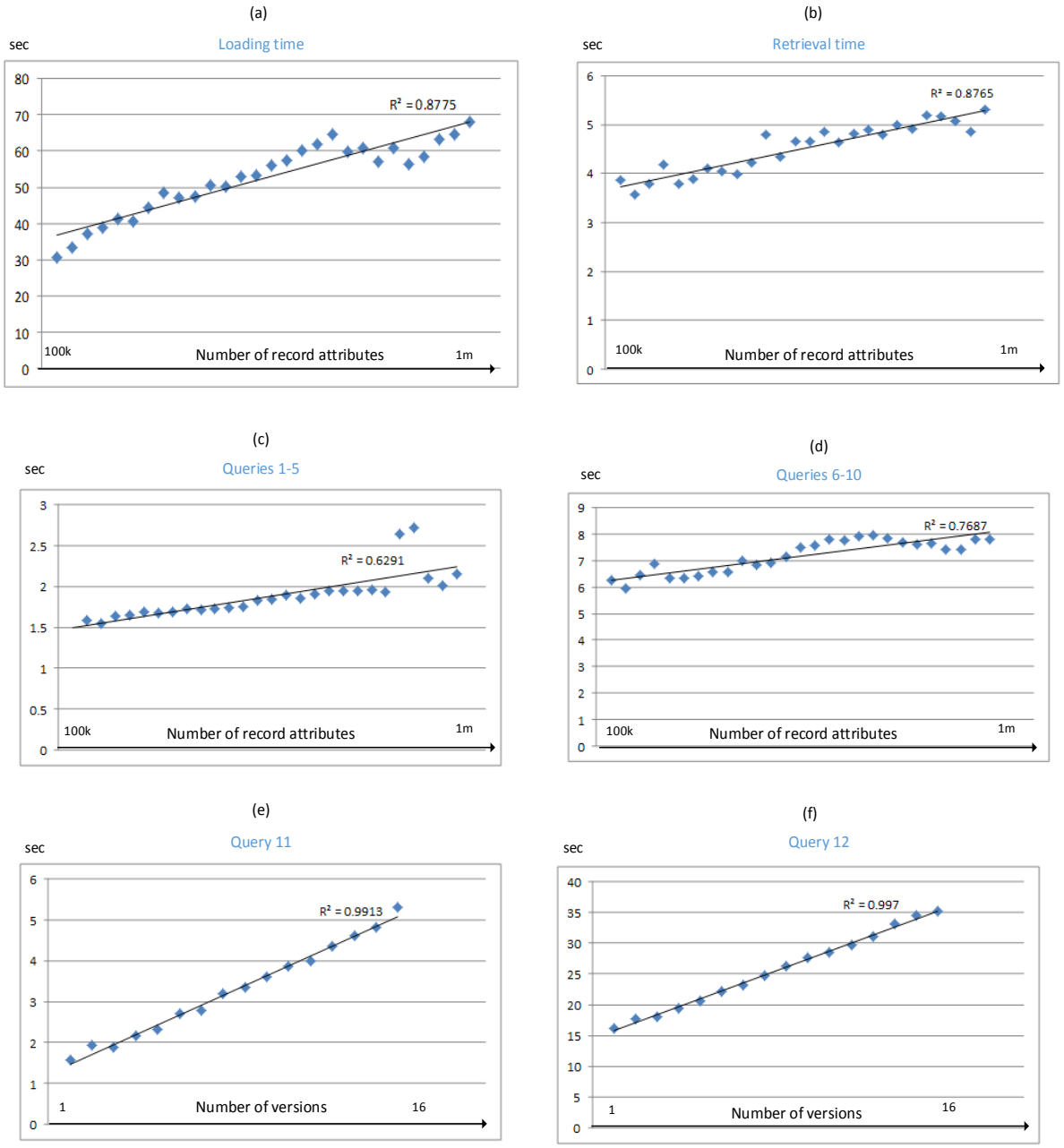

Figure 6: Loading times (a), retrieval times (b), select queries without filters and aggregates (c), select queries with filters and aggregates (d), select queries with variable datasets (e)-(f).

6 Related Work

Managing LOD evolution is a multi-faceted problem that consists in versioning, efficient archiving, change representation and detection, model abstraction and provenance issues, among others. Work has been done in most of these fields individually, but few approaches have regarded the issue as a singular problem of many interdependencies, less so in the case of the Data Web, where datasets evolve independently, often in non-centralized ways, while citing and using one another. Versioning for LOD in the context of complete systems or frameworks has been addressed in [3][8][13][14][18][21]. Ontology or schema based approaches have been proposed in [17][19][20] with the most prominent example being the PAV ontology [20], a specialization of the W3C recommended PROV ontology [22] for modelling provenance.

As far as querying is concerned, work has been done in extending SPARQL with temporal capabilities [27][28][29][30][31. In [27] no data model is proposed, instead temporal information is used to separate triples in different named graphs. Incorporation of annotations and provenance on the query side has been approached in [29] where triple annotations serve as context and an extension of SPARQL is proposed. In [30] an ontology-based approach is followed where temporal reasoning capabilities are provided to OWL-2.0 and SPARQL is extended to cater for the temporal dimension. In [31] a triple store is implemented that incorporates spatiotemporal querying by utilizing the SPARQL extensions proposed in [28].

In [6] an approach is presented that builds on the Memento [7] framework, an extension of HTTP to include a traversable and queryable temporal dimension, adapted for LOD purposes. Non-changing, time-independent URIs are employed for current state identification. Dereferencing past versions of resources is done with temporal content negotiation, an HTTP extension.

In [8], the authors tackle the problem of web versioning by providing extended functionality to the web server. They focus on web documents and components (HTML, images etc.). They associate versions with 'transaction times' and they perform the archiving process only when a web document is requested from the server. This creates a distinction between known and assumed past versions, making the whole process lossy and not consistent with realistic expectations for LD archiving. However, the system is not burdened with constant change tracking and makes a reasonable assumption that versions that have never been accessed are perhaps of no significant importance as far as archiving is concerned.

In [9], the authors tackle the problem of version management for XML documents by using deltas to capture differences between sequential versions and use deltas as edit scripts to yield sequential versions. The introduced space redundancy is compensated by the query efficiency of storing complete deltas rather than compressed deltas. They go on to define change detection as the computation of non-empty deltas and they argue that past version retrieval can be achieved by storing all complete deltas as well as a number of complete intermediate versions, finding the bounding versions of the desired ones and applying their corresponding deltas. Finally, they use a query language based on XQuery in order to enable longitudinal querying and they provide tag indices for each edit operation for faster delta application.

In [10], the authors propose a method for archiving scientific data from XML documents. The approach targets individual elements in the DOM tree of an XML document, rather than the whole versions themselves. They use time stamping in order to differentiate between the states of a particular element in different time intervals and they store each element only once in the archive. The timestamps are pushed down to the children of an element in order to reflect the changes at the corresponding level of the tree, an approach also followed in [11].

In [12] the authors study the change frequency of LOD sources and the implications on dataset dynamics. They differentiate between the document-centric and the entity-centric perspectives of change dynamics, the latter further divided into the entity-per-document and global entity notions. We partially adopt this distinction in our work, as will be described further on.

SemVersion [13] computes the semantic differences as well as the structural differences between versions of the same graph but is limited to RDFS expressiveness. DSNotify [14] is an approach to deal with dataset dynamics in distributed LD. The authors identify several levels for the requirements of change dynamics, namely, vocabularies for describing dynamics, vocabularies for representing changes, protocols for change propagation and algorithms and applications for change detection. It implements a change detection framework which incorporates these points in a unified functionality scheme, having as main motivation the problem of link maintenance. When dealing with changes, they target the what, how, when and why dimensions of the changes, closely related to the problem of provenance. They differentiate between triple and resource level for the *what* dimension and they argue that the level selection depends on the particular use case. How is expressed by the differential operators associated with a change (such as add/remove or compound changes) while when is expressed by timestamps and version numbers. Finally, the why dimension is usually expressed in manual annotations.

These approaches do not address evolution as a multi-faceted problem. Our approach differentiates itself by considering versioning, annotating, change management and dataset heterogeneity as necessary components of evolution and are thus tackled together. Furthermore, most of the work presented in this section address the temporal aspect of evolution in datasets, instead we chose to consider temporality as an inherent characteristic of versioning. It is trivial to explicitly create temporal operators for DIACHRON QL by evaluating datasets over their temporal metadata and translating temporal operators to version-based operators such as `AT VERSION` or `BETWEEN VERSIONS`.

7 Conclusions

In this paper, we have discussed the challenges and requirements for the preservation and evolution management of datasets published on the Data Web and we have presented an archiving approach that utilizes a novel conceptual model and query language for storing and querying evolving heterogeneous datasets and their metadata. The DIACHRON data model and QL have been applied to real world datasets from

the life-sciences and open government statistical data domains. An archive that employs these ideas has been implemented and its performance has been tested using real versions of datasets from the aforementioned domains over a series of loading, retrieval and querying operations.

The growing availability of open linked datasets has brought forth significant new problems related to the distributed nature and decentralized evolution of LOD and has posed the need for novel efficient solutions for dealing with these problems. In this respect, we have highlighted some possible directions and presented our work that tackles evolution and captures several dimensions regarding the management of evolving information resources on the Data Web.

8 References

1 C. Bizer, T. Heath, T. Berners-Lee: Linked Data – The Story So Far. Special. Issue on Linked Data, In International Journal on Semantic Web and Information Systems, 2009

2 J. Umbrich, M. Hausenblas, A. Hogan, A. Polleres, S. Decker: Towards Dataset Dynamics: Change Frequency of Linked Open Data Sources. In LDOW 2010

3 M. Meimaris, G. Papastefanatos, C. Pateritsas, T. Galani, and Y. Stavrakas. Towards a Framework for Managing Evolving Information Resources on the Data Web. In PROFILES2014.

4 Stavrakas, Yannis, George Papastefanatos, Theodore Dalamagas, and Vassilis Christophides. "Diachronic Linked Data: Towards Long-Term Preservation of Structured Interrelated Information." *arXiv preprint arXiv:1205.2292* (2012).

5 Manyika, James, Michael Chui, Brad Brown, Jacques Bughin, Richard Dobbs, Charles Roxburgh, and Angela H. Byers. "Big data: The next frontier for innovation, competition, and productivity." (2011)

6 Van de Sompel, Herbert, Robert Sanderson, Michael L. Nelson, Lyudmila L. Balakireva, Harihar Shankar, and Scott Ainsworth. "An HTTP-based versioning mechanism for linked data." *arXiv preprint arXiv:1003.3661* (2010).

7 Van de Sompel, Herbert, Michael L. Nelson, Robert Sanderson, Lyudmila L. Balakireva, Scott Ainsworth, and Harihar Shankar. "Memento: Time travel for the web." *arXiv preprint arXiv:0911.1112* (2009).

8 Dyreson, Curtis E., Hui-Ling Lin, and Yingxia Wang. "Managing versions of web documents in a transaction-time web server." In *Proceedings of the 13th international conference on World Wide Web*, pp. 422-432. ACM, 2004.

9 Wong, Raymond K., and Nicole Lam. "Managing and querying multi-version XML data with update logging." In *Proceedings of the 2002 ACM symposium on Document engineering*, pp. 74-81. ACM, 2002.

10 Buneman, Peter, Sanjeev Khanna, Keishi Tajima, and Wang-Chiew Tan. "Archiving scientific data." *ACM Transactions on Database Systems (TODS)*29, no. 1 (2004): 2-42.

11 Papastefanatos, George, Yannis Stavrakas, and Theodora Galani. "Capturing the history and change structure of evolving data." In *DBKDA 2013, The Fifth International Conference on Advances in Databases, Knowledge, and Data Applications*, pp. 235-241. 2013.

12 Umbrich, Jürgen, Michael Hausenblas, Aidan Hogan, Axel Polleres, and Stefan Decker. "Towards dataset dynamics: Change frequency of linked open data sources." (2010).

13 Völkel, Max, and Tudor Groza. "SemVersion: An RDF-based ontology versioning system." In *Proceedings of the IADIS international conference WWW/Internet*, vol. 2006, p. 44. 2006.

14 Popitsch, Niko P., and Bernhard Haslhofer. "DSNotify: handling broken links in the web of data." In *Proceedings of the 19th international conference on World wide web*, pp. 761-770. ACM, 2010.

15 Umbrich, Jürgen, Boris Villazón-Terrazas, and Michael Hausenblas. "Dataset dynamics compendium: A comparative study." (2010).

16 Papavassiliou, Vicky, Giorgos Flouris, Irini Fundulaki, Dimitris Kotzinos, and Vassilis Christophides. *On detecting high-level changes in RDF/S KBs*. Springer Berlin Heidelberg, 2009.

17 Grandi, F.. Light-weight Ontology Versioning with Multi-temporal RDF Schema. In *SEMAPRO 2011, The Fifth International Conference on Advances in Semantic Processing* (pp. 42-48), 2011.

18 Im, D. H., Lee, S. W., & Kim, H. J. (2012). A version management framework for RDF triple stores. *International Journal of Software Engineering and Knowledge Engineering*, *22*(01), 85-106.

19 Keivanloo, Iman, Christopher Forbes, Juergen Rilling, and Philippe Charland. "Towards sharing source code facts using linked data." In Proceedings of the 3rd International Workshop on Search-Driven Development: Users, Infrastructure, Tools, and Evaluation, pp. 25-28. ACM, 2011.

20 Ciccarese, Paolo, Stian Soiland-Reyes, Khalid Belhajjame, Alasdair JG Gray, Carole A. Goble, and Tim Clark. "PAV ontology: provenance, authoring and versioning." J. Biomedical Semantics 4 (2013): 37.

21 Klein, Michel, Dieter Fensel, Atanas Kiryakov, and Damyan Ognyanov. "Ontology versioning and change detection on the web." In Knowledge Engineering and Knowledge Management: Ontologies and the Semantic Web, pp. 197-212. Springer Berlin Heidelberg, 2002.

22 Lebo, Timothy, Satya Sahoo, Deborah McGuinness, Khalid Belhajjame, James Cheney, David Corsar, Daniel Garijo, Stian Soiland-Reyes, Stephan Zednik, and Jun Zhao. "Prov-o: The prov ontology." W3C Recommendation, 30th April (2013).

23 Stefanidis, Kostas, Ioannis Chrysakis, and Giorgos Flouris. "On Designing Archiving Policies for Evolving RDF Datasets on the Web." In *Conceptual Modeling*, pp. 43-56. Springer International Publishing, 2014.

24 Papastefanatos, George. Challenges and Opportunities in the Evolving Data web..1st International Workshop on Modeling and Management of Big Data (MoBiD), November 13, 2013, Hong Kong, 2013.

25 Malone, James, Ele Holloway, Tomasz Adamusiak, Misha Kapushesky, Jie Zheng, Nikolay Kolesnikov, Anna Zhukova, Alvis Brazma, and Helen Parkinson. "Modeling sample variables with an Experimental Factor Ontology."*Bioinformatics* 26, no. 8 (2010): 1112-1118.

26 Meimaris, Marios, Chrysakis, Ioannis, Flouris, Giorgos, Galani, Theodora, Hasapis Panagiotis, Papastefanatos, George, Pateritsas Christos, Stavrakas, Yannis and Stefanidis, Kostas. DIACHRON Archiving Structures and Associated Metadata. Deliverable 4.1 of project FP7-601043 (DIACHRON). March 2013.

27 Tappolet, Jonas, and Abraham Bernstein. "Applied temporal RDF: Efficient temporal querying of RDF data with SPARQL." In *The Semantic Web: Research and Applications*, pp. 308-322. Springer Berlin Heidelberg, 2009.

28 Perry, Matthew, Prateek Jain, and Amit P. Sheth. "Sparql-st: Extending sparql to support spatiotemporal queries." In *Geospatial semantics and the semantic web*, pp. 61-86. Springer US, 2011.

29 Lopes, Nuno, Axel Polleres, Umberto Straccia, and Antoine Zimmermann. "AnQL: SPARQLing up annotated RDFS." In *The Semantic Web–ISWC 2010*, pp. 518-533. Springer Berlin Heidelberg, 2010.

30 Batsakis, Sotiris, Kostas Stravoskoufos, and Euripides GM Petrakis. "Temporal reasoning for supporting temporal queries in OWL 2.0." In *Knowledge-Based and Intelligent Information and Engineering Systems*, pp. 558-567. Springer Berlin Heidelberg, 2011.

31 Kyzirakos, Kostis, Manos Karpathiotakis, and Manolis Koubarakis. "Strabon: a semantic geospatial DBMS." In *The Semantic Web–ISWC 2012*, pp. 295-311. Springer Berlin Heidelberg, 2012.