GeoTriples: Transforming Geospatial Data into RDF Graphs Using R2RML and RML Mappings

Kostis Kyzirakos[a], Dimitrianos Savva[b], Ioannis Vlachopoulos[b], Alexandros Vasileiou[b], Nikolaos Karalis[b], Manolis Koubarakis[b], Stefan Manegold[a]

[a]*Database Architectures Group, Centrum Wiskunde & Informatica, Amsterdam, The Netherlands*
[b]*Dept. of Informatics and Telecommunications, National and Kapodistrian University of Athens, University Campus, Ilissia, Athens 15784, Greece*

Abstract

A lot of geospatial data has become available at no charge in many countries recently. Geospatial data that is currently made available by government agencies usually do not follow the linked data paradigm. In the few cases where government agencies do follow the linked data paradigm (e.g., Ordnance Survey in the United Kingdom), specialized scripts have been used for transforming geospatial data into RDF. In this paper we present the open source tool GeoTriples which generates and processes extended R2RML and RML mappings that transform geospatial data from many input formats into RDF. GeoTriples allows the transformation of geospatial data stored in raw files (shapefiles, CSV, KML, XML, GML and GeoJSON) and spatially-enabled RDBMS (PostGIS and MonetDB) into RDF graphs using well-known vocabularies like GeoSPARQL and stSPARQL, but without being tightly coupled to a specific vocabulary. GeoTriples has been developed in European projects LEO and Melodies and has been used to transform many geospatial data sources into linked data. We study the performance of GeoTriples experimentally using large publicly available geospatial datasets, and show that GeoTriples is very efficient and scalable especially when its mapping processor is implemented using Apache Hadoop.

1. Introduction

In the last few years, the area of linked geospatial data has received attention as researchers and practitioners have started tapping the wealth of existing geospatial information and making it available on the Web [20, 21]. As a result, the linked open data (LOD) cloud has been slowly populated with geospatial data. For example, Great Britain's national mapping agency, Ordnance Survey, has been the first national mapping agency that has made various kinds of geospatial data from Great Britain available as linked open data[1]. Similarly, projects TELEIOS[2], LEO[3], MELODIES[4] and Copernicus App Lab[5], in which our research groups participated, published a number of geospatial datasets that are Earth observation products e.g., CORINE Land Cover and Urban Atlas[6]. Also, the Spatial Data on the Web working group[7] created jointly by the Open Geospatial Consortium (OGC) and the World Wide Web Consortium (W3C) has produced in 2017 five relevant working notes on best practices, use cases and requirements, Earth observation data, spatio-temporal data cubes and coverages as linked data.

Geospatial data can come in vector or raster form and are usually accompanied by metadata. Vector data, available in formats such as ESRI shapefiles, KML, and GeoJSON documents, can be accessed either directly or via Web Services such as the OGC Web Feature Service or the query language of a geospatial DBMS. Raster data, available in formats such as GeoTIFF, Network Common Data Form (netCDF) and Hierarchical Data Format (HDF), can be accessed either directly or via Web Services such as the OGC Web Coverage Processing Service (WCS) or the query language of an array DBMS, e.g., rasdaman[8] or MonetDB/SciQL. Metadata about geospatial data are encoded in various formats ranging from custom XML schemas to do-

[1] `http://data.ordnancesurvey.co.uk/`
[2] `http://www.earthobservatory.eu/`
[3] `http://www.linkedeodata.eu/`
[4] `https://www.melodiesproject.eu/`
[5] `https://www.app-lab.eu/`
[6] `http://kr.di.uoa.gr/#datasets`
[7] `https://www.w3.org/2015/spatial/wiki/Main_Page`
[8] `http://www.rasdaman.org/`

main specific standards like the OGC GML Application schema for EO products and the OGC Metadata Profile of Observations and Measurements. Automating the process of transforming input geospatial data to linked data has only been addressed by few works so far [3, 11, 23, 15, 26]. In many cases, for example in the wildfire monitoring and management application that we developed in TELEIOS [23], custom Python scripts were used for transforming all the necessary geospatial data into linked data.

In this paper we extend the mapping languages R2RML[9] and RML[10] with some new constructs that help to specify ways of transforming geospatial data from its original format into RDF. We also present the tool GeoTriples that generates automatically and processes extended R2RML and RML mappings for transforming geospatial data from various formats into RDF graphs. The input formats supported are spatially-enabled relational databases (PostGIS and MonetDB), ESRI shapefiles, XML documents following a given schema (hence GML documents as well), KML documents, JSON and GeoJSON documents and CSV documents. GeoTriples is a semi-automated tool that enables the automatic transformation of geospatial data into RDF graphs using state of the art vocabularies like GeoSPARQL [2], but at the same time it is not tightly coupled to a specific vocabulary. The transformation process comprises three steps. First, GeoTriples generates automatically extended R2RML or RML mappings for transforming data that reside in spatially-enabled databases or raw files into RDF. As an optional second step, the user may revise these mappings according to her needs e.g., to utilize a different vocabulary. Finally, GeoTriples processes these mappings and produces an RDF graph.

Users can store and query an RDF graph generated by GeoTriples using a geospatial RDF store like Strabon[11]. They can also interlink this graph with other linked geospatial data using tools like the temporal and geospatial extension of Silk[12] developed in our group [30] or the more recent tool Radon developed with the participation of our group [29]. For example, it might be useful to infer links involving topological relationships e.g., `A geo:sfContains F` where `A` is the area covered by a remotely sensed multispectral image `I`, `F` is a geographical feature of interest (field, lake, city etc.) and `geo:sfContains` is a topological relationship from the topology vocabulary extension of GeoSPARQL. The existence of this link might indicate that `I` is an appropriate image for studying certain properties of `F`.

It is often the case in applications that relevant geospatial data is stored in spatially-enabled relational databases (e.g., PostGIS) or files (e.g., shapefiles), and its owners do not want to explicitly transform it into linked data [7, 10]. For example, this might be because these data sources get frequently updated and/or are very large. If this is the case, GeoTriples is still very useful. GeoTriple users can use the generated mappings in the system Ontop-spatial to view their data sources *virtually* as linked data. Ontop-spatial is a geospatial extension of the Ontology-Based Data Access (OBDA) system Ontop[13] developed by our group [4]. Ontop performs on-the-fly SPARQL-to-SQL translation on top of relational databases using ontologies and mappings. Ontop-spatial extends Ontop by enabling on-the-fly GeoSPARQL-to-SQL translation on top of geospatial databases. The experimental evaluation of [4] has shown that this approach is not only simpler for the users as it does not require transformation of data, but also more efficient in terms of query response time.

GeoTriples is an open source tool that has been developed in the context of the EU FP7 projects LEO and MELODIES mentioned in the beginning of this section. It is currently utilized in the EU Horizon 2020 project Copernicus App Lab where data from three Copernicus Services[14] (Land, Marine and Atmosphere) are made available as linked data to aid their take-up by mobile developers.

The organization of the paper is as follows. Section 2 presents background information and discusses related work. In Section 3 we present the extensions to the mapping languages R2RML and RML for the geospatial domain. In Section 4 we present the architecture of GeoTriples and discuss how GeoTriples generates automatically mappings, and how these mappings are subsequently processed for transforming a geospatial data source into an RDF graph. Section 5 gives an example of translating an input shapefile into RDF, using the GeoTriples utilities. Section 6 presents an implementation of the mapping process of GeoTriples that uses Apache Hadoop. In Section 7 we perform a performance evaluation of the implementations of GeoTriples using publicly available geospatial data. We also compare GeoTriples with the similar tool TripleGeo. Finally, in Section 8, we conclude the paper and discuss future work.

[9] `https://www.w3.org/TR/r2rml/`
[10] `http://rml.io/`
[11] `http://www.strabon.di.uoa.gr/`
[12] `http://silk.di.uoa.gr/`
[13] `http://ontop-spatial.di.uoa.gr/`
[14] `http://www.copernicus.eu/`

2. Background and Related Work

In this section we present related work on methodologies and tools for transformation of data sources into RDF graphs. Currently, most similar approaches have been focusing on mapping relational databases into RDF graphs. We will discuss two state-of-the-art approaches, direct mapping and R2RML and a recent proposal for mapping heterogeneous data into RDF, the mapping language RML. We also include related work on transforming geospatial data into RDF graphs based on these mapping techniques.

2.1. Direct Mapping of Relational Data to RDF

A straightforward mechanism for mapping relational data into RDF is the *direct mapping* approach that became a W3C recommendation in 2012 [9]. In this approach tables in a relational database are mapped to classes defined by an RDFS vocabulary, while attributes of each table are mapped to RDF properties that represent the relation between subject and object resources. Identifiers, class names, properties, and instances are generated automatically following the respective labels of the input data. For example, given the table `Address`, the class `<Address>` is generated, and every tuple is represented by a resource that becomes an instance of this class. The generation of RDF data is dictated by the schema of the relational database. This mechanism was initially defined in [8], and [32] is an implementation of it.

2.2. The Mapping Language R2RML

A language for expressing customized mappings from relational databases to RDF graphs is the *R2RML mapping language* that became W3C recommendation in 2012 [14]. R2RML mappings provide the user with the ability to express the desired transformation of existing relational data into the RDF data model, following a structure and a target vocabulary that is chosen by him or her. R2RML mappings refer to logical tables to retrieve data from an input database. A *logical table* can be a relational table, an SQL view that exists in a database or an SQL SELECT query. A triples map is defined for each logical table that will be exported into RDF. A *triples map* is a rule that defines how each tuple of the logical table will be mapped to a set of RDF triples. A triples map consists of a subject map and one or more predicate-object maps. A *subject map* is a rule that defines how to generate the URI that will be the subject of each generated RDF triple. Usually, the primary key of the relation is used for this purpose. A *predicate-object* map consists of predicate maps and object maps. A *predicate map* defines the RDF property to be used to relate the subject and the object of the generated triple. An *object map* defines how to generate the object of the triple, the value of which originates from the value of the attribute of the specified logical table. Subject maps, predicate maps and object maps are term maps. A *term map* is a function that generates an RDF term from a logical table. Three types of term maps are defined: constant-valued term maps that always generate the same RDF term, column-valued term maps that generate RDF terms from an attribute of the input relation, and template-valued term maps that generate RDF terms according to a template. R2RML defines the vocabulary to express foreign key relationships among logical tables. For this purpose, a join condition is introduced for defining the column name of the child table and the column name of the parent table. Figure 1a presents an overview of R2RML.

Features of R2RML. R2RML is not limited to mapping relational tables to RDFS classes and relational attributes to data properties. R2RML has several other features that are presented below:

- *Ad-hoc SQL result sets*: This feature is useful in cases where the user wants to apply some transformations (e.g., syntactic modifications) or apply aggregate functions on the input data.
- *Templates*: Using the `rr:template` property, one can specify the format of a resource that will be used as a subject or an object of a triple using a string template. For example, consider the relational table *Employee*(*id*, *name*, *surname*, *salary*). The subject of the generated resource could use the primary key *id* of the table to form a resource URI template `"http://example.com/Employee/{id}/"` to generate automatically resources of the form `<http://example.com/Employee/1/>`, `<http://example.com/Employee/2/>`, etc.
- *Linking two tables*: Most RDF datasets do not use only data properties (properties for which the value is a data literal), but also object properties (properties for which the value is an individual) to assert relations between resources. As a result, an R2RML mapping can take into account foreign key constraints that may exist in the underlying relational database to make such assertions.
- *Named Graphs*: Named graphs are a key concept of RDF that allows the identification of an RDF graph using a URI. As a result, contextual information like

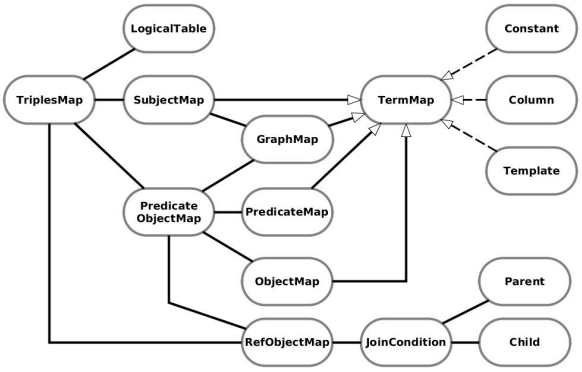
provenance information, can be naturally expressed in RDF. R2RML allows a user to customize a subject map so that produced triples can belong to the default graph or any other named graph.

2.3. The Mapping Language RML

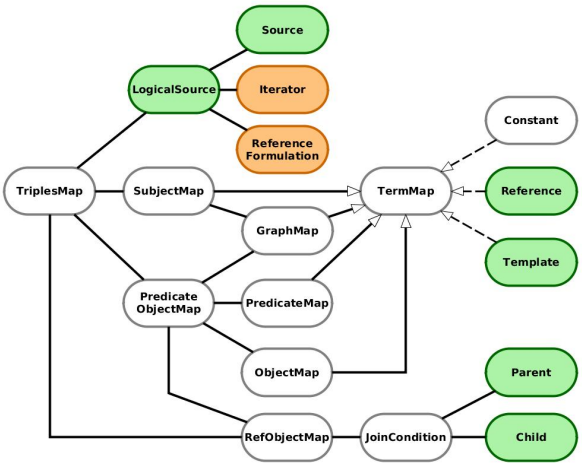
The RDF Mapping language (RML) [18, 17] is a recently proposed generic mapping language which can express rules that map data with heterogeneous structures and serializations to RDF graphs. RML is defined as a superset of R2RML and allows the expression of rules that map relational and semi-structured data (e.g., XML, JSON) into RDF graphs. The main feature of RML is that it provides the vocabulary for defining a generic data source and the iterator pattern over the input data. Note that R2RML does not define explicitly an iterator pattern over the input data since a per row iteration is implied. In contrast, RML allows the user to explicitly define an iterator that defines how the source data should be accessed. For example, an XPath expression can be defined as an iterator over an XML document, a JSONPath expression can be defined as an iterator over a JSON document and an SQL query can be defined as an iterator over a relational database. Figure 1b presents an overview of RML.

RML extensions to R2RML. RML has redefined all classes and properties defined in R2RML that are strictly coupled to the relational model as follows:

- The concept of `logical table` has been replaced by the concept of `logical source` which is a more generic concept that covers many kinds of input data sources. A logical source contains all necessary properties for accessing a data source and iterating over it. Similarly, the concept of `table` has been replaced by the more general concept of `source` which is a pointer to a dataset.
- The concept of `iterator` is a new concept that instructs a processor on how to access data from a logical source. The iterator is accompanied by a `referenceFormulation` property that specifies the query language that is being used by it. For example, for transforming an XML document into RDF, we can set the `referenceFormulation` to be the XPath language and the `iterator` to be the XPath query itself. Currently, the following reference formulations are defined: `rr:sqlQuery`, `ql:CSV`, `ql:XPath`, `ql:CSS3` and `ql:JSONPath`.
- The `column` property has been replaced by the more general `reference` property. This property is used to point to the data that is being returned by the iterator.



(a) R2RML overview



(b) RML overview

Figure 1: R2RML and RML overview. White boxes denote R2RML components, green boxes denote R2RML components extended by RML and orange boxes denote RML specific components. Arrows with white arrowhead denote subclasses, arrows with dashed line and white arrowhead denote the different types of TermMap and simple lines denote associations.

2.4. Transforming Geospatial Data into RDF

Recently, enough attention has been paid to the problem of making geospatial data available on the Web as linked data. In many cases linked geospatial datasets are either generated manually or by semi-automated processes from original data sources such as shapefiles or spatially-enabled relational databases. On the contrary, a plethora of tools are currently available for publishing relational and non-relational data as linked data. These tools may follow the direct mapping approach, may support a mapping language, may support relational or non-relational data and may be able to evaluate SPARQL queries by translating them into SQL queries.

The project LinkedGeoData[15] [3, 31] focuses on publishing OpenStreetMap[16] data as linked data. In this context the tool Sparqlify[17] has been developed and used. Sparqlify is a SPARQL to SQL rewriter which allows one to define RDF views over a relational database and query them using SPARQL. Sparqlify uses the Sparqlification mapping language that has similar expressivity with R2RML but different syntax. Sparqlify supports some basic geospatial capabilities, like handling the serializations of a geometry and evaluating topological predicates like the function `st_intersects` that returns whether two geometries share some portion of the space.

The tool Geometry2RDF[18] [15] was the first tool that allowed the user to convert geospatial information that resides in a spatially-enabled relational database into an RDF graph. Geometry2RDF takes as input data stored in a spatially-enabled relational DBMS and utilizes the libraries Jena and GeoTools to produce an RDF graph. Geometry2RDF follows the direct mapping approach, allows the user to configure the properties that connect a URI to the serialization of a geometry and allows for the conversion of the coordinates to the desired coordinate reference system. Geometry2RDF is no longer maintained by its developers (Oscar Corcho, private communication). The codebase of Geometry2RDF was the basis of the first version of tool TripleGeo which is discussed below.

An interesting approach appears in [11] where the authors present how R2RML can be combined with a spatially-enabled relational database in order to transform geospatial data into RDF. For the manipulation of the geometric information prior to its transformation into RDF, the authors create several logical tables that are based on ad-hoc SQL queries that perform the appropriate pre-processing (e.g., requesting the serialization of a geometry according to the WKT standard). This approach demonstrates the power of utilizing a general-purpose mapping language like R2RML in the case of geospatial data. However, in [11] , no automated method for transforming geospatial datasets into RDF is discussed, and dealing with different types of data formats (e.g., shapefiles) was not considered.

The tool TripleGeo has been developed in the context of European FP7 project GeoKnow[19] [26]. TripleGeo is the closest existing tool to GeoTriples (let alone the similarity in name). TripleGeo can extract and transform geospatial features from many input formats: relational DBMSs via JDBC (PostgreSQL/PostGIS, Oracle Spatial and Graph, MySQL and MS SQL Server) and raw files (ESRI shapefiles, GeoJSON, GML, KML, GPX and CSV). TripleGeo consists of three modes: (i) the GRAPH mode, which transforms the input dataset into an RDF graph, (ii) the STREAM mode, in which each entry of the input data is processed separately, and (iii) the RML mode, which uses RML mappings for the conversion of the data. Modes STREAM and GRAPH are able to transform only up to four attributes of each tuple. These attributes are the ID, the geometry, the name and the category. This feature limits the user from extracting other useful information that may exist in a data source. Thanks to its modular implementation, TripleGeo is now being enhanced by its developers with more utilities without affecting existing functionality (Spiros Athanasiou, personal communication). In the context of the SLIPO project[20], it is planned to further extend TripleGeo with several novel features, and most importantly, specific functionalities that can efficiently support transformation of large datasets of points of interest (POIs). Further, it is planned to include support for de facto POI formats (like TomTom Overlay, OziExplorer Waypoints etc.), more DBMS platforms (e.g., SpatialLite), as well as direct access to OpenStreetMap data files. Finally, TripleGeo already supports RDF transformation from INSPIRE metadata as well as certain INSPIRE data themes (Geographical names, Administrative units, Addresses, Cadastral parcels, Transport networks, Hydrography, Protected sites).

Recently, the OBDA engine Ontop[21] [28] has been extended in Ontop-spatial [4]. Ontop-spatial is a framework for OBDA enriched with geospatial functionality. It supports the evaluation of stSPARQL/GeoSPARQL queries over virtual RDF graphs defined through R2RML mappings to a relational database. It is a mature system that has already been used in a number of applications [7, 10]. Handling geometric information in raw files (e.g., shapefiles) or made available through the scientific data access service OPeNDAP has been added to Ontop-spatial [5] by integrating the relational engine madIS [12]. This is work done in the context of project Copernicus App Lab which has been discussed in the introduction.

Oracle has recently implemented in its well-known DBMS many interesting features for linked geospatial

[15] `http://linkedgeodata.org/`

[16] `http://www.openstreetmap.org/`

[17] `http://aksw.org/Projects/Sparqlify.html`

[18] `http://mayor2.dia.fi.upm.es/oeg-upm/index.php/en/technologies/151-geometry2rdf/`

[19] `https://github.com/SLIPO-EU/TripleGeo`

[20] `http://www.slipo.eu/`

[21] `http://ontop.inf.unibz.it/`

data. First of all, it has offered support for GeoSPARQL in Oracle 12c, Release 1. Recently, they have also implemented support for RDF Views of relational tables with SDO_GEOMETRY columns. This feature is available in Oracle 12c Release 2[22]. Any SDO_GEOMETRY columns in the mapped relational tables can be exposed as geo:wktLiteral and GeoSPARQL queries against the RDF views will utilize any spatial indexes that have been created on the underlying relational tables. The virtual RDF can be queried in SQL with SEM_MATCH or through their Joseki/Fuseki-based SPARQL endpoint. The recent Oracle presentation "Realizing the Benefits of Linked Geospatial Data with R2RML and GeoSPARQL" at the most recent SmartData conference[23] gives details of these approaches (Matthew Perry, personal communication).

3. Extending the Mapping Languages R2RML and RML for Geospatial Data

Much work has been done recently on extending RDF to represent and query geospatial information. The most mature results of this work are the data model stRDF and the query language stSPARQL [24, 6] and the OGC standard GeoSPARQL [2]. These data models and query languages have been implemented in many geospatial triple stores including Strabon, GraphDB[24], Oracle Spatial and Graph[25], etc.

stRDF is an extension of the W3C standard RDF that allows the representation of geospatial data that changes over time [24, 6]. stRDF is accompanied by stSPARQL, an extension of the query language SPARQL 1.1 for querying and updating stRDF data. stRDF and stSPARQL use OGC standards WKT and GML for a serialized representation of temporal and geospatial data.

GeoSPARQL is an OGC standard for the representation and querying of linked geospatial data. GeoSPARQL defines much of what is required for such a query language by providing a vocabulary (classes, properties, and functions) that can be used in geospatial RDF graphs and SPARQL queries. The top level classes defined in GeoSPARQL are geo:SpatialObject the instances of which include everything that can have a spatial representation, and geo:Feature that represents all features and is the superclass of all classes of features that the users might want to define. To represent geometric objects, the class geo:Geometry is introduced. The topology vocabulary extension of GeoSPARQL provides a vocabulary for asserting and querying topological relations between spatial objects. The extension is parameterized by the family of topological relations supported. Such relations can be the ones defined in the OGC standard for simple features [1] (e.g., geo:sfEquals), the Egenhofer relations [19] (e.g., geo:ehMeet), or the RCC-8 relations [27] (e.g., geo:rcc8ec). These relations can be asserted in a triple of an RDF graph (e.g., ex:Athens geo:sfWithin ex:Greece .) or can be used in a triple pattern of a SPARQL query (e.g., ?x geo:sfWithin ex:Greece).

When transforming geospatial data into RDF graphs using a vocabulary like the vocabulary of stRDF or GeoSPARQL, we may need to compute on the fly values that are not explicitly present in the source data such as the dimension of a given geometry, the length of a line or the area of a polygon. Such values can be derived by applying a transformation function over the input geometries. In addition, we may want to compute on the fly which topological, directional, or distance relations hold between two spatial objects. Such values can be derived by evaluating a topological, directional, or distance function over the input geometries. As a result, we need to extend the R2RML and RML mapping language with new classes and properties in order to allow the representation of such *transformation functions*. This is presented in detail in the rest of this section. The new prefix that we introduce for our constructs is rrx for http://geotriples.di.uoa.gr/ns/rml_extensions.

3.1. Tranformation Functions for R2RML and RML

We introduce two new properties as extensions to the R2RML language. The first property is rrx:function and it is used for representing transformation functions. The value of a rrx:function property is an IRI that identifies a SPARQL extension function that performs a desired transformation. The domain of the object property rrx:function is an rr:TermMap and the range of this property is an rrx:TransformationFunction.

We also define the property rrx:argumentMap for representing an ordered sequence of term maps that will be passed as arguments to a transformation function. The domain of the object property rrx:argumentMap is an rr:TermMap. The rrx:argumentMap property

[22] http://docs.oracle.com/database/122/RDFRM/rdf-views.htm#RDFRM555

[23] http://smartdata2017.dataversity.net/sessionPop.cfm?confid=110&proposalid=9947

[24] https://ontotext.com/products/graphdb/

[25] http://www.oracle.com/technetwork/database/options/spatialandgraph/overview/index.html

has as range an `rdf:List` of term maps that define the arguments to be passed to the transformation function.

The following definition extends the concepts of a term map so that transformation functions can be represented.

Definition 1. *A* transformation-valued term map *is a term map that generates an RDF term by applying a SPARQL extension function on one or more term maps. A transformation-valued term map has exactly one* `rrx:function` *property and one* `rrx:argumentMap` *property.*

Definition 2. *A* term map *must be a constant-valued term map, a column-valued term map, a template-valued term map, or a transformation-valued term map depending on what properties are being used.*

Example 1. *The following is an object map that is a transformation-valued term map:*

```
rr:objectMap [ rrx:function strdf:dimension ;
               rrx:argumentMap (
                [ rr:column "Geom" ] ); ] .
```

The above map defines that the generated RDF triples will have as objects the RDF terms that result from applying the SPARQL extension function `strdf:dimension` *to the values of the column* `Geom`*.*

Example 2. *The following is an object map that is a transformation-valued term map that has a transformation function that takes multiple arguments as input:*

```
rr:objectMap [ rrx:function geof:buffer ;
               rrx:argumentMap (
                   [ rr:column "Geom" ]
                   [ rr:constant "10";
                    rr:datatype xsd:int ]
                   [ rr:constant uom:metre
                    ] ) ] .
```

The above map instructs that the generated RDF triples will have as objects new geometric objects that represent all points whose distance from the geometries stored in the `Geom` *column is less than or equal to ten meters.*

In R2RML, a referencing object map is used for representing foreign key relationships among logical tables. A referencing object map may contain one or more join conditions that define the child and parent columns of the foreign key. Two tuples are considered as qualified when their values for the corresponding child and parent columns are equal. For allowing the usage of a different predicate, we need to extend the definition of a referencing object map. This need arises from the topology vocabulary of GeoSPARQL that allows the user to assert that a topological relation holds between two geometric objects. In order to generate datasets that explicitly contain qualitative topological information, we need to extend the definition of referencing object maps and join conditions.

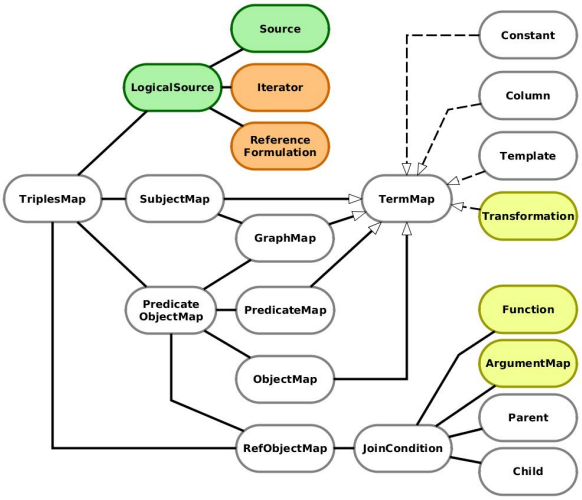


Figure 2: Overview of the extensions to R2RML and RML. White boxes denote R2RML components, green boxes denote R2RML components extended by RML, orange boxes denote RML specific components and yellow boxes denote our extensions. Arrows with white arrowhead denote subclasses, arrows with dashed line and white arrowhead denote the different types of TermMap and simple lines denote associations.

Definition 3. *A* join condition *is a resource that:*

- *has exactly one value for the* `rr:child` *and* `rr:parent` *properties, or*
- *has exactly one value for the* `rrx:function` *and* `rrx:argumentMap` *properties. Each element of the argument map optionally has a* `rr:triplesMap` *property in order to define the triples map from where the joining values derive. If a* `rr:parentTriplesMap` *is absent, the term map is evaluated over the current triples map.*

Example 3. *The following is an R2RML join condition:*

```
rr:joinCondition [
    rrx:function geof:sfOverlaps;
    rrx:argumentMap ( [ rr:column "Geom" ]
                      [ rr:column "Geom" ;
                        rr:triplesMap <MapB> ])
                       ] .
```

The join condition above states that the values of the `Geom` *column of the current triples map must spatially overlap with the values of the* `Geom` *column from the triples map* `<MapB>`.

Definition 4. *A* referencing object map *is a map that allows a predicate-object map to generate as objects the subjects of another triples map. A referencing object map can be represented as a resource that:*

- *has exactly one* `rr:parentTriplesMap` *property and optionally one or more join conditions, or*
- *has at least one join condition that employs one transformation function.*

Example 4. *The following is an R2RML referencing object map:*

```
rr:objectMap [
   rr:joinCondition [
      rrx:function ex:isClose;
      rrx:argumentMap([rr:column "Geom"]
                      [rr:column "Geom";
                       rr:triplesMap <MapB>]
                      [rr:constant "10";
                      rr:datatype xsd:int ]
                      [rr:constant uom:metre
                       ] ) ] ] .
```

Let `ex:isClose` be a SPARQL extension function which takes as input two geometries, a decimal number and a URI that denotes a unit of measurement, and returns true if the distance between the two geometries is less than the given decimal number in the given unit of measurement. Then, the values of the above object map are the subjects from the triples map `MapB` that correspond to a geometry that is close to the geometries of the current triples map.

Given that the extensions that we defined above are orthogonal to the RML extensions of R2RML, the same extensions can be viewed as extensions of RML. In Figure 2 we give a graphical overview of R2RML, RML and our extensions.

4. The Tool GeoTriples

In this section we present the tool GeoTriples that we developed for transforming geospatial data sources into RDF. GeoTriples[26] is an open-source tool that is distributed freely according to the Mozilla Public License v2.0. We will present the architecture of GeoTriples and discuss its main components and their respective implementation details. We will then describe how GeoTriples generates R2RML and RML mappings for transforming data that reside in spatially-enabled databases and raw files, and, finally, and how it processes these mappings to produce an RDF graph that follows the GeoSPARQL,stRDF or any other user-defined vocabulary.

4.1. System Architecture

In this section we present the system architecture of GeoTriples that is depicted in Figure 3. The input data for GeoTriples can be geospatial data and metadata stored in ESRI Shapefiles, XML, GML, KML, JSON, GeoJSON and CSV documents or spatially-enabled relational databases (e.g., PostGIS and MonetDB). GeoTriples has a connector that is responsible for providing an abstraction layer that allows the rest of the components to transparently access the input data. GeoTriples comprises three main components: the mapping generator, the mapping processor and the stSPARQL/GeoSPARQL evaluator.

The *mapping generator* is given as input a data source and creates automatically an R2RML/RML mapping document, depending on the type of the input. The generated mapping is enriched with subject and predicate-object maps, taking into account all transformations that are needed to produce an RDF graph that is compliant with the GeoSPARQL vocabulary. Afterwards, the user may edit the generated mapping document to make it comply with her requirements (e.g., use a different vocabulary). We point out that the ability of GeoTriples to use different vocabularies is a useful feature since even standardized vocabularies such as the one of GeoSPARQL can be dropped, modified or extended in the future.

The *mapping processor* may use either the generated mapping document or one created by the user from scratch. Based on the triples map definitions in the mapping file, the component generates the final RDF graph which can be manifested in any of the popular RDF syntaxes such as Turtle, RDF/XML, Notation3 or N-Triples. The mapping processor has been implemented in two ways. The first implementation runs on a single processor, while the second runs in a distributed manner using the Apache Hadoop framework. The second implementation will be described in detail in Section 6.

The *stSPARQL/GeoSPARQL evaluator* is a component that evaluates an stSPARQL/GeoSPARQL query over a relational database given an R2RML mapping. The evaluator is a thin layer that integrates GeoTriples with the OBDA engine Ontop-spatial [4]. It supports

[26] `http://geotriples.di.uoa.gr`

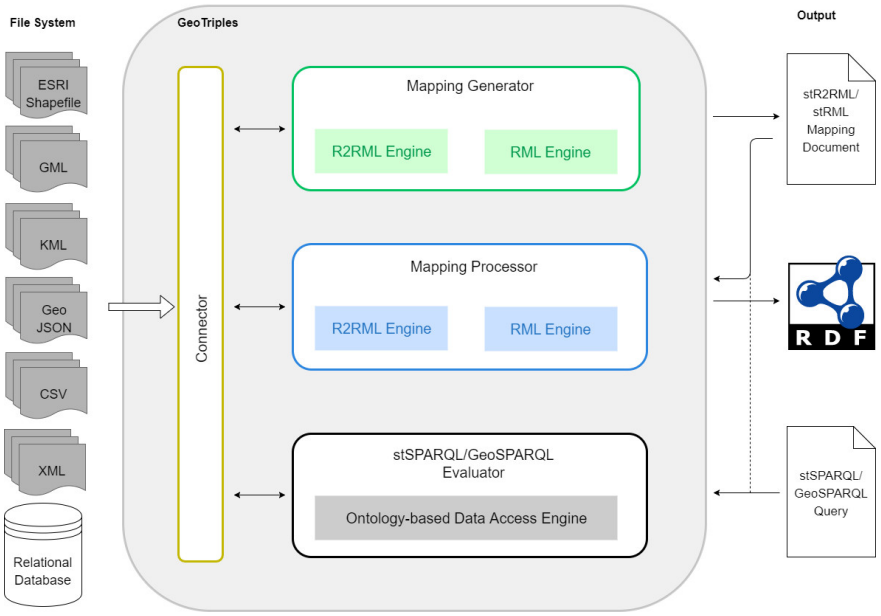


Figure 3: The system architecture of GeoTriples

the evaluation of stSPARQL/GeoSPARQL queries over virtual RDF graphs defined through R2RML mappings to a geospatial relational database.

4.2. Implementation Details

To implement GeoTriples, we chose to extend the D2RQ platform [13] which is a well-known system for publishing relational data into RDF. D2RQ provides an interface for generating and processing R2RML mappings for a variety of relational databases that are accessible via JDBC[27]. To support the processing of other data sources, GeoTriples also extends the iMinds RML processor[28] to process RML mappings of relational databases as well as other data sources. GeoTriples uses the GeoTools[29] library for processing geometric objects within ESRI shapefiles, GML, KML, GeoJSON and CSV documents. The GDAL[30] library is also integrated in the application as an alternative for processing ESRI shapefiles more efficiently.

4.3. Automatic Generation of R2RML and RML Mappings

GeoTriples can automatically produce an R2RML or RML mapping document that can then be used to generate an RDF graph that corresponds to the input database or file.

Let us first discuss how mappings are generated when the input is a geospatial relational database or a shapefile. In the next section, we also present a simple example that illustrates the functionality of GeoTriples. In these cases, R2RML/RML mappings that are generated by GeoTriples consist of two triples maps: one for handling thematic information and one for handling geospatial information. The triples map that handles non-geometric (thematic) information defines a logical table that contains the attributes of the input data source and a unique identifier for the generated instances. The latter could be either the primary key of the table or a row number of each tuple in the dBase table of a shapefile.[31] Combined with a URI template, the unique identifier is used to produce the URIs that are the subjects of the produced triples. For each column of the input data source, GeoTriples generates an RDF predicate according to the name of the column and a predicate object map. This map generates predicate-object pairs consisting of the generated predicate and the column values.

The triples map that handles geospatial information defines a logical table with a unique identifier similar to the thematic one. The logical table contains a serialization of the geometric information according to the WKT format, and all attributes of the geometry that

[27] `https://docs.oracle.com/javase/8/docs/technotes/guides/jdbc/`

[28] `http://rml.io/`

[29] `http://geotools.org/`

[30] `http://www.gdal.org/`

[31] A short description of what a shapefile is is given in Section 5.

are required for producing a GeoSPARQL compliant RDF graph. For this purpose, if the input is a shapefile, GeoTriples constructs RML mappings with transformations that invoke GeoSPARQL/stSPARQL extension functions. If the input is a relational database, GeoTriples constructs SQL queries that utilize the appropriate spatial functions of the Open Geospatial Consortium standard "Simple Feature Access - Part 2: SQL Option"[32] that generate the information required. For example, in order to generate triples that describe the dimensionality of a geometry, GeoTriples creates an RML mapping that invokes the `strdf:dimension` SPARQL extension function that is evaluated over a geometry of a shapefile, or utilizes the PostGIS SQL function `ST_Dimension` when dealing with a spatially-enabled relational database.

A different approach is followed when the input data source is an XML document. In this case, GeoTriples utilizes information from the XML Schema Definition (XSD) language file that describes the structure of the input XML document for generating the appropriate RML mappings. In XML Schema, two kinds of types are defined: simple and complex. Simple types[33], whether built-in (e.g., `xsd:integer`) or user-defined, are restrictions of the base type definitions (e.g., positive integers with maximum value 122 for representing age information). We define as *simple element* an XML element of simple type. By definition, a simple element may contain only text and cannot contain any other XML elements or attributes. A complex type[34] is a set of attribute declarations and a content type that is applicable to the attributes and children of an element information item respectively. We define a *complex element* to be an XML element of complex type.

For mapping an XSD document to an ontology, we define a strategy mapping XSD elements and attributes to RDFS classes and properties. We introduce three rules for the generation of an RML mapping and an ontology for any XML schema.

1. Each simple element is mapped to a predicate-object map and an OWL data type property.
2. Each complex element is mapped to a triples map and an RDFS class.
3. Nested complex elements are mapped to a predicate-object map and an OWL object property.

A well-adopted standard for representing geospatial features in XML is the Geography Markup Language (GML) which has been defined by the Open Geospatial Consortium. We introduce the following rules for handling geometric information that may be present in an input XSD document that follows the GML standard.

1. For every `gml:AbstractGeometryType`, we create a new triples map. All generated IRIs will be instances of the class `ogc:Geometry`.
2. For each newly created triples map, we generate a predicate-object map for each geometric property defined in the geometry extension component of GeoSPARQL. Each predicate-object map must utilize the appropriate GeoSPARQL or stSPARQL extension functions for performing the desired transformation.
3. We generate a property-object map that will create a `ogc:hasGeometry` link between the current triples map and the triples map of the parent XML element.

Besides the data model used by GML and KML documents, there is no other standard practice to represent geospatial information inside XML documents. As a result, a custom approached must be followed by editing the RML mapping accordingly.

In Table 1 we present the transformation functions that we implemented in GeoTriples. In the Strabon website[35], one may find a complete reference of these functions offered by stSPARQL.

4.4. Processing of R2RML and RML Mappings

In this section we will present in detail the R2RML and RML processor of GeoTriples. The algorithms employed by the processor are Algorithm 1 and 2 where we highlight with comments our extensions. In Algorithm 1 we present how GeoTriples processes an R2RML mapping when the input data source is a spatially-enabled relational database. In this case GeoTriples uses the extended D2RQ mapping processor. The process starts by parsing the input mapping and storing it in memory. For each triples map, the mapping processor performs the following steps. At first, the processor extracts the logical table from the document, constructs the effective SQL query, and stores it in memory. Every logical table has an *effective SQL query* that, if

[32] `http://www.opengeospatial.org/standards/sfs`

[33] `http://www.w3.org/TR/xmlschema11-1/#Simple_Type_Definition`

[34] `http://www.w3.org/TR/xmlschema11-1/#Complex_Type_Definition`

[35] `http://www.strabon.di.uoa.gr/files/stSPARQL_tutorial.pdf`

stSPARQL Functions	GeoSPARQL Functions	Description
strdf:dimension	geo:dimension	Returns the inherent dimension of the input geometry.
strdf:spatialDimension	geo:spatialDimension	Returns the dimension of the spatial portion of the input geometry. If the spatial portion does not have a measure coordinate, this will be equal to the coordinate dimension (see below).
strdf:coordinateDimension	geo:coordinateDimension	Returns the number of measurements or axes needed to describe the position of this geometry in its coordinate system.
strdf:isEmpty	geo:isEmpty	Returns true if the input geometry is an empty geometry. If true, then this geometry represents an empty geometry collection, polygon, point etc.
strdf:isSimple	geo:isSimple	Returns true if the input geometry has no anomalous geometric points, such as self-intersection or self-tangency.
	geo:is3D	Returns true if the geometry uses three spatial dimensions.
strdf:asText	geo:asWKT	Returns the Well-Known Text (WKT) serialization of the input geometry.
strdf:asGML	geo:asGML	Returns the Geography Markup Language (GML) serialization of the input geometry.

Table 1: Transformation functions supported by GeoTriples

executed, produces as its result the contents of the logical table. If the subject map is a template-valued term map or a column-valued term map, the related columns are extracted and stored in memory. Then, the processor iterates over all predicate-object maps, and for each one it extracts all template- and column-valued term maps. These term maps are cached in memory along with the position that they appear on (i.e., whether they are a subject, predicate, object or graph map). Notice that there is no upper bound on the number of predicate or object maps that may appear in a predicate-object map. Afterwards, the processor constructs an SQL query statement that projects all column names that are referenced by the term maps that appear in the subject, predicate and object positions for the current predicate map. The constructed query is posed to the database and then the processor iterates over the results. For each predicate and object value in the result row, a new RDF triple is constructed. If the object map is a referencing object map, a new SQL query is constructed. The `SELECT` clause will contain the column names that are referenced by the subject map of the parent triples map and the subject and predicate column names of the current predicate-object map. The effective SQL queries of the current triples map and the parent triples map are used as the relations in the `FROM` clause. The child and parent columns are joined in the `WHERE` clause of the query. If there are more than one referencing object maps in the same predicate object map, the `WHERE` clause will contain multiple equi-joins between the child and parent column names.

For processing RML mappings, the GeoTriples mapping processor extends the RML processor of the tool iMinds. In Algorithm 2 we present how GeoTriples processes an RML mapping. For each triples map, it opens the data source defined in the logical source and poses the defined iterator query to the data source, using the appropriate library. After receiving the result set, the mapping processor iterates through all features in the results, and for each feature it iterates through all predicate-object maps and processes each one to form the desired RDF triples. For each feature, the processor extracts the values that are referenced by template-valued and reference-valued term maps that appear in the current predicate-object map. In the case of a referencing object map, the processor accesses the logical source of the parent triples map, to get the resulting features. Then, it selects only the features that have equal

Algorithm 1 Processing R2RML mappings

```
1: Data: R2RML Mapping /* The mapping can also contain transformation functions */
2: Result: RDF graph
3: for each triples map in mapping do
4:     Scan logical table;
5:     effectiveQuery ← ConstructEffectiveQuery(logical table);
6:     sColumns ← ExtractColumnNames(subject map);
7:     /* We have extended predicate-object map, in order to support transformation functions */
8:     for each predicate-object map in triples map do
9:         for each predicate map in predicate-object map do
10:            pColumns ← ExtractColumnNames(predicate map);
11:            for each object map do
12:                if ObjectMapType(object map) = referencing object map then
13:                    parentTriplesMap ← GetParentTriplesMap(object map);
14:                    parentEffectiveQuery ← GetParentEffectiveQuery(parentTriplesMap);
15:                    parentTMColumns ← XtractColNamesFromParentSubject(parentTriplesMap);
16:                    childColumn ← GetChildColumn(object map);
17:                    parentColumn ← GetParentColumn(object map);
18:                    effectiveQuery ← ConstructJointEffectiveQuery(effectiveQuery,
19:                    parentEffectiveQuery, childColumn, parentColumn);
20:                    projections ← sColumns, pColumns, parentTMColumns;
21:                else
22:                    oColumns ← ExtractColumnNames(object map);
23:                    projections ← sColumns, pColumns, oColumns;
24:                end if
25:                resultSet ← PoseQuery(projections, effectiveQuery);
26:                for each result row in resultSet do
27:                    /* We have extended the process of the construction of the RDF triples,
28:                    in order to produce the results of the transformation functions */
29:                    ConstructRDFTriple(result row);
30:                end for
31:            end for
32:        end for
33:    end for
34: end for
```

Algorithm 2 Processing RML mappings

```
1: Data: RML Mapping /* The mapping can also contain transformation functions */
2: Result: RDF graph
3: for each triples map in mapping do
4:     Scan logical source;
5:     iterator ← ExtractIterator(logical source);
6:     ReferenceFormulation ← ExtractReferenceFormulation(logical source);
7:     logicalReferences ← ExtractLogicalReferences(triples map);
8:     subjectMap ← ExtractSubjectMap(triples map);
9:     switch ReferenceFormulation do Select processor implementation
10:        case Xpath Select XML processor;
11:        case JSONPath Select JSON processor;
12:        case SHP Select Shapefile processor;
13:        case SQL Select SQL processor;
14:        default Error(Wrong input)
15:        /* The iterator is an SQL query which projects all logical references */
16:        iterator ← ConstructNewSQLIterator(logicalReferences, iterator);
17:    endsw
18:    resultSet ← ExecuteIterator(iterator);
19:    for each result row in resultSet do
20:        sValues ← ExtractSubjectValues(SubjectMap, result row);
21:        /* We have extended predicate-object map, in order to support transformation functions */
22:        for each predicate-object map do
23:            for each predicate map do
24:                pValues ← ExtractPredicateValues(predicate map, result row);
25:                for each object map do
26:                    if ObjectMapType(object map) = referencing object map then
27:                        parentTriplesMap ← GetParentTriplesMap(object map);
28:                        parentSubjectMap ← ExtractSubjectMap(parentTriplesMap);
29:                        childColumn ← GetChildReference(object map);
30:                        childValue ← ExtractChildValue(object map, result row);
31:                        parentColumn ← GetParentReference(object map);
32:                        parentResultSet ← ExtractResultSet(parentTriplesMap);
33:                        for each p result row in parentResultSet do
34:                            sParentValues ← ExtractSubjectValues(parentSubjectMap, result row);
35:                            parentValue ← ExtractParentValue(object map, result row);
36:                            if childValue = parentValue then
37:                                /* We have extended the process of the construction of the RDF triples,
38:                                in order to produce the results of the transformation functions */
39:                                ConstructRDFTriple(sValues, pValues, sParentValues);
40:                            end if
41:                        end for
42:                    else
43:                        oValues ← ExtractObjectValues(object map, result row);
44:                        /* Our extension takes place here as well */
45:                        ConstructRDFTriple(sValues, pValues, oValues);
46:                    end if
47:                end for
48:            end for
49:        end for
50:    end for
51: end for
```

values on the parent and the child references. For these features, an RDF triple is generated using the result of the parent triples map's subject map as the object RDF term. The same procedure is followed for each referencing object map that may appear in the RML mapping.

5. An Example

Let us now show an example of RML mapping generation by GeoTriples for an input shapefile.

A *shapefile* is a vector data storage format for storing the location, shape, and attributes of geographic features. It is an open specification which has been developed by ESRI in the context of its ArcGIS product. Shapefiles can represent geographic features along with the spatial and non-spatial attributes that describe them. For example, they can store the geometry of a country in conjunction with its name, population etc. An ESRI shapefile dataset is a collection of files stored in the same directory. Three important files are the ones with the suffixes `.shp`, `.dbf` and `.shx`. The `.shp` file is the main file that contains the geometry of one or more features, the `.dbf` file contains the non-spatial (thematic) attributes of these features in a table with dBASE format, and the `.shx` file is a positional index of the feature geometry to allow seeking forwards and backwards quickly.

In our example, we will use a shapefile containing information about the country Italy from the database of Global Administrative Areas (GADM). GADM is a geospatial database of the world's administrative areas which are countries and lower level subdivisions such as provinces, states etc.[36] A subset of the thematic information for the feature "Italy", from the corresponding `.dbf` file, is presented in Table 2. A subset of the geometric information for the same feature, from the `.shp` file, is presented in Table 3.

As shown in Table 2, the `.dbf` file contains a unique identifier `SHAPE_ID` and the thematic attributes of the identified features (name of the country, name of the region, its type etc.)

As shown in Table 3, the `.shp` file contains a unique identifier `SHAPE_ID` and the coordinates X and Y of all points forming the polygons of the identified features. Unique identifiers in the `.shp` file correspond to unique identifiers in the `.dbf` file and establish the identity of the features described by the two files.

Given Italy's shapefile as input, GeoTriples will generate a corresponding RML mapping file, parts of which will be presented and explained immediately. The mapping file consists of a thematic part and a geometry part as we discussed in Section 4.3 above.

First, the thematic part contains information about the logical source, the type of the file and the iterator of the file:

```
rml:logicalSource [ rml:source
"User/data/ITA_adm_shp/ITA_adm1.shp";
                    rml:referenceFormulation ql:SHP;
                    rml:iterator "ITA_adm1"; ];
```

Subsequently, the triples map of the data source is given. This starts with the subject map with a URI which is generated by a template which includes a unique identifier `GeoTriplesID`:

```
rr:subjectMap [ rr:template
   "http://linkedeodata.eu/ITA_adm1/id/{GeoTriplesID}";
               rr:class onto:ITA_adm1; ];
```

The default namespace of the predicates of the generated triples is `http://linkedeodata.eu/ontology#` and its prefix is `onto`. Then the predicate-object maps are given. For reasons of brevity, we give only the maps for the `ISO` and the `NAME_1` thematic attributes. The `ISO` attribute gives rise to the predicate `onto:hasISO` and the `NAME_1` attribute gives rise to the predicate `onto:hasNAME_1`:

```
rr:predicateObjectMap [
          rr:predicateMap [ rr:constant onto:hasISO ];
          rr:objectMap [ rr:datatype  xsd:string;
                         rml:reference "ISO"; ]; ];

rr:predicateObjectMap [
          rr:predicateMap [ rr:constant onto:hasNAME_1 ];
          rr:objectMap [ rr:datatype  xsd:string;
                         rml:reference "NAME_1";  ]; ];
```

The geometry part is structured like the thematic part. First, it contains information about the logical source, the type of the file and the iterator of the file:

```
rml:logicalSource [ rml:source
"User/data/ITA_adm_shp/ITA_adm1.shp";
                    rml:referenceFormulation ql:SHP;
                    rml:iterator "ITA_adm1"; ];
```

Then the triples map of the data source is given starting with the subject map. The corresponding URI is generated by a template which includes the unique identifier `GeoTriplesID`. This URI can be given as input to GeoTriples. The subject map makes this URI an instance of the class `ogc:Geometry` of the GeoSPARQL ontology:

```
rr:subjectMap [
         rr:template
         "http://linkedeodata.eu/ITA_adm1/
          Geometry/{GeoTriplesID}";
         rr:class ogc:Geometry; ];
```

[36] `http://www.gadm.org/`

SHAPE_ID	ID_0	ISO	NAME_0	ID_1	NAME_1	CCA_1	TYPE_1	ENGTYPE_1	VARNAME_1
7.0	112	ITA	Italy	8	Lazio	12	Regione	Region	Lacio/Latium
14.15	112	ITA	Italy	15	Sicily	19	Regione	Region	Sicilia

Table 2: Thematic information from the `.dbf` file

SHAPE_ID	X	Y
7.0	13.4551401138	40.792640686
7.0	13.4551401138	40.7923622131
7.0	13.4556941986	40.7923622131
...	...	...
14.15	12.4334716797	37.8940315247
14.15	12.4334716797	37.8937492371
14.15	12.4323616028	37.8937492371
...	...	...
19.2	12.5093050003	44.9306945801
19.2	12.5093050003	44.9304161072
19.2	12.5095834732	44.9304161072
...	...	...

Table 3: Geometric information from the `.shp` file

Finally, the geometry part contains the predicate-object maps. These are generated by transformation functions computed on the geometry, for example the predicates `geo:dimension` and `geo:asWKT` of the GeoSPARQL ontology:

```
rr:predicateObjectMap [
            rr:predicateMap [ rr:constant geo:dimension ];
            rr:objectMap [ rr:datatype  xsd:integer;
                 rrx:function geo:dimension;
                 rrx:argumentMap ( [ rml:reference
                      "the_geom"; ] ); ]; ];

rr:predicateObjectMap [
            rr:predicateMap [ rr:constant geo:asWKT ];
            rr:objectMap [ rr:datatype  ogc:wktLiteral;
                 rrx:function geo:asWKT;
                 rrx:argumentMap ( [ rml:reference
                           "the_geom"; ] ); ]; ];
```

The last step of the operation of GeoTriples is the processing of the generated R2RML and RML mappings to produce an output RDF graph. As the algorithm of Section 4.4 dictates, initially, for each triples map, we extract the logical source of the file, the reference formulation to choose the right processor (e.g., shapefile processor), the subject map and the corresponding iterator. Then, for each element of the logical source, the iterator extracts the subject value of the generated RDF triple. Then the predicate-object maps generate the predicate-object pairs of the triple. For the triple map presented above, three of the generated thematic triples for feature "Lazio" are:

```
PREFIX geo: <http://www.opengis.net/ont/geosparql#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX adm: <http://linkedeodata.eu/ITA_adm1/>
PREFIX onto: <http://linkedeodata.eu/ontology#>
PREFIX admgeo: http://linkedeodata.eu/ITA_adm1/Geometry/>
PREFIX leo: <http://linkedeodata.eu/ontology#>

adm:8 rdf:type leo:ITA_adm1 .
adm:8 onto:hasNAME_1 "Lazio"^xsd:string .
adm:8 geo:hasGeometry admgeo:8 .
```

In the same way, the geometric part of the mapping file generates, among others, the following triples corresponding to the geometric attributes of the same feature:

```
admgeo:8 rdf:type geo:Geometry .
admgeo:8 geoisEmpty "false"^^xsd:boolean .
admgeo:8 geois3D "false"^^xsd:boolean .
admgeo:8 geo:isSimple "true"^^xsd:boolean .
admgeo:8 geo:coordinateDimension "9644"^^xsd:integer .
admgeo:8 geo:dimension "2"^^xsd:integer .
admgeo:8 geo:asWKT
"<http://www.opengis.net/def/crs/EPSG/0/4326>
MULTIPOLYGON (((13.455140113830566 40.79264068603521,
 13.455140113830566 40.79236221313482, ...,
12.455550193786678 41.90755081176758)))"^^geo:wktLiteral .
admgeo:8 geo:spatialDimension "2"^^xsd:integer .
```

6. Implementing the Mapping Processor of GeoTriples Using Apache Hadoop

To enable the efficient transformation of large or numerous input geospatial files into RDF, we have developed an implementation of the GeoTriples mapping processor using Apache Hadoop.[37] We call this implementation *GeoTriples-Hadoop* and present its architecture in Figure 4. Apache Hadoop is an open source framework that allows the distributed processing of large datasets across clusters of computers. The main components of Apache Hadoop are HDFS (its distributed file system) and Hadoop MapReduce (an implementation of the MapReduce programming model originally introduced by Google [16]). We have implemented the mapping processor for the case of RML mappings generated from shapefiles and CSV files. Our implementation is freely available on GitHub like the single-node implementation discussed above.[38]

The mapping processor of GeoTriples-Hadoop is implemented by mappers in the MapReduce programming

[37] http://hadoop.apache.org/

[38] https://github.com/dimitrianos/GeoTriples-Hadoop

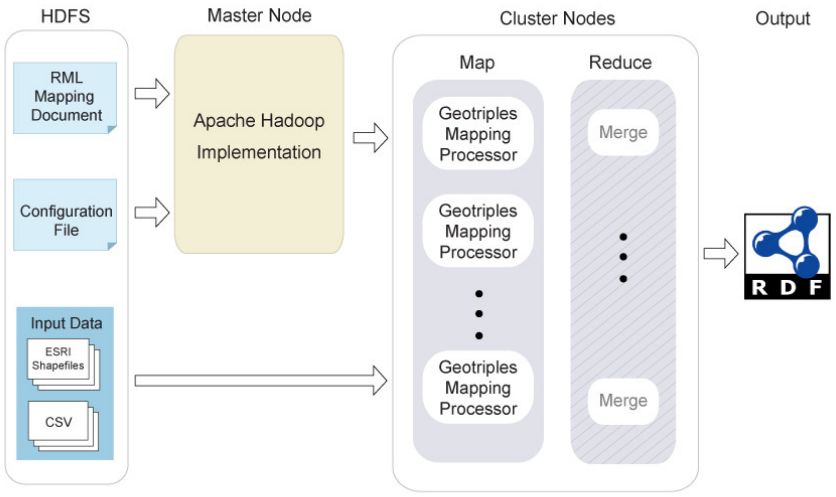


Figure 4: The system architecture of GeoTriples-Hadoop

model. Each mapper takes as input one shapefile or a block of a CSV file and produces one RDF file as output. The use of reducers is optional: they can be used for the merging of the RDF files produced by the mappers. For example, if we have 100 mappers and 2 reducers, the mappers will create 100 RDF files and the reducers will merge the results into 2 RDF files. For the processing of shapefiles by Hadoop, we used the open source library Shapefile[39]. Shapefile is a very efficient and lightweight Java library that contains classes that enable Hadoop to read shapefiles that are stored in the HDFS.

To be able to use the Shapefile library effectively, we had to solve an incompatibility with GeoTriples and deal with one drawback. The incompatibility with GeoTriples stems from the fact that Shapefile is based on the ESRI Geometry API[40] while GeoTriples is based on the JTS Topology Suite[41]. To solve this incompatibility, we had to change the way in which Shapefile process the geometries. In addition, we made an improvement in the processing of shapefiles by creating a hybrid library class that can process both geometry types (points and polygons) in the same execution. The original library had two different classes, one for shapefiles that contain points and one for shapefiles that contain polygons something that is inconvenient when using the library. Finally, we converted the Shapefile library into a Maven project.[42] In this way, the GeoTriples implementation that uses Hadoop is a Maven project that consists of three completely independent modules: the module that contains the Apache Hadoop implementation, the module that contains the rest of the components of the GeoTriples tool discussed above and the module that contains the Shapefile library.

The main advantage of the GeoTriples-Hadoop implementation of the mapping processor is the distribution of the transformation workload to clusters of computing nodes. It is well-known that an Apache Hadoop implementation is very efficient only with large datasets. Thus, the single-node implementation of the mapping processor will typically be more efficient than the Hadoop implementation for smaller datasets when we take into account the costs for the initialization and the management of the Hadoop cluster.

The mapping processor of GeoTriples-Hadoop uses the Shapefile library to distribute the workload by assigning each one of the shapefiles of the input dataset to a different mapper. This might appear to be contrary to the Hadoop principle of segmenting each input file according to the blocksize, and distributing the segments to the cluster nodes where the mappers reside.

[39] `https://github.com/mraad/Shapefile`
[40] `https://github.com/Esri/geometry-api-java`
[41] `https://github.com/locationtech/jts`

[42] Apache Maven is a software project management tool that helps Java software developers manage the software development process. For more, see `https://maven.apache.org/`.

The Shapefile library does not support this principle; instead, it uses a different map procedure for accessing a whole shapefile. In practice this is not a drawback of the Shapefile library (and our implementation) because, typically, the average size of a shapefile is smaller than the typical size of an Apache Hadoop blocksize, typically 64MB-128MB (see for example the average size of a shapefile in the datasets of Table 4). Most shapefiles we have encountered in our work are tens of MBs in size. Fewer shapefiles are in the order of hundreds of MBs, and very few are 1GB or more. In fact, according to ESRI[43], each component of a shapefile cannot exceed 2GBs in size.

In the case of CSV files, since CSV file access is built-in in Apache Hadoop, the Hadoop principle of segmenting an input file according to blocksize and distributing it to mappers is also followed by our implementation.

7. Performance Evaluation of GeoTriples

In this section we present a performance evaluation of three versions of GeoTriples: the single-node implemenation (called simply GeoTriples in this section), the GeoTriples-Hadoop implementation, and a version of the single-node implementation which uses the shell tool GNU Parallel[44] and multiple threads to parallelize the work of processing the mappings (called *GeoTriples-Multi* in this section). For a fairer comparison of GeoTriples-Hadoop and GeoTriples-Multi, we choose the number of threads made available to GeoTriples by GNU Parallel to be equal to the number of the Hadoop cluster nodes in GeoTriples-Hadoop (15 threads for 15 cluster nodes). We also present the results of the comparison of GeoTriples with the similar tool TripleGeo. TripleGeo has already been described in Section 2.4.

For evaluating the performance of the various implementations, we used Earth observation data from the Sentinel Open Access Hub managed by the European Space Agency, Dutch cartographic data, an urban land use dataset made available by the European Environmental Agency and shapefiles from the Global Administrative Areas portal. The kinds of input formats we used are spatially-enabled relational databases (PostGIS and MonetDB), shapefiles and CSV files. We first evaluate GeoTriples exhaustively and then we compare it with GeoTriples-Hadoop, GeoTriples-Multi and TripleGeo. For all evaluations, we start by discussing some measurement assumptions that we adopted in our study, then we define the experimental platform that was used for carrying out the experiments, and, finally, we present and discuss our findings.

7.1. Measurement Assumptions

In the experiments with the implementation of GeoTriples, we focus on the time required for generating and processing R2RML and RML documents. The index creation for shapefiles, the database loading, and indexing is beyond the scope of the experiments. The rationale is based on the predominantly read-only nature of RDF stores.

The timing for generating the whole RDF graph focuses on cold runs. Cold run is a run of the query right after which all caches of the operating system are cleared, the DBMS is re-started and no data is loaded into the system's main memory, neither by the DBMS, nor in file system caches.

Elapsed time is the real time required for performing all necessary steps for transforming a shapefile or the corresponding relational table, into an RDF graph stored as a file on disk. This includes the cost of accessing the shapefile or accessing the database for requesting exactly the same information (this includes the time required for parsing, optimizing, executing a query and transferring the results to the client).

The computations carried out by GeoTriples are I/O and CPU intensive. The I/O intensitivity reveals itself mostly when there are many and large files in the input data, and this has as result many I/O transactions. The CPU intensitivity reveals itself when the input data contains large geometries and transformation functions are computed on them on the fly.

7.2. Experimental Setup

Our experiments were carried out on a Fedora 20 (Linux 3.12.10) installation on an Intel Core i7-2600K with 8 MB cache running at 3.4 GHz (turbo 3.8 GHz). The CPU has four cores and each core has two threads. The system has 16GB of RAM and a 2 TB disk with 32MB cache and rotational speed is 7200 rpm. The I/O read speed is 110-115 MB/s.

7.3. Datasets for the First Set of Experiments

We transformed into RDF three datasets: the metadata of all Sentinel-2A Earth Observation products, the Dutch TOP10NL cartographic dataset and the Urban

[43] `http://support.esri.com/technical-article/000010813`

[44] `https://www.gnu.org/software/parallel/`

Atlas dataset of the European Environmental Agency. Details for each dataset are presented in Table 4.

Sentinel-2[45] is a European wide-swath, high-resolution, multi-spectral imaging mission. For our first set of experiments, we used the metadata of all Level-1C products that are currently available from the Sentinel Open Access Hub[46]. Level-1C products contain information about top-of-atmosphere reflectances in cartographic geometry. The dataset consists of 1,950 GML files and 234 XML files. The size of the source GML and XML files is 3.4 MB and 183 MB respectively. For transforming this dataset into RDF, we used GeoTriples for generating and processing the appropriate RML mappings.

TOP10NL[47] is a nationwide digital topographic data base published by the Dutch Cadastre, Land Registry and Mapping Agency. It is the most detailed product of the Dutch digital topographic data base. The dataset consists of 85 GML files with information about the Dutch road network, railway network, etc. and 16 XML files that contain versioning information for the TOP10NL features. The size of the source GML and XML files is 26 GB and 5 GB respectively. For transforming this dataset into RDF, we used GeoTriples for generating and processing the appropriate RML mappings.

Urban Atlas[48] is an activity of the European Environment Agency that provides reliable, inter-comparable, high-resolution land use maps for 336 Large Urban Zones and their surroundings (more than 100,000 inhabitants as defined by the Urban Audit) for the reference year 2006. The Urban atlas dataset is distributed as 336 ZIP files. After decompressing the ZIP files, the complete dataset occupies approximately 20 GB of space and contain information about 5,762,507 features within 336 shapefiles. Since domain experts usually store this dataset into a spatially-enabled relational database, we also stored all shapefiles into MonetDB and PostgreSQL enhanced with the PostGIS extension. For transforming this dataset into RDF, we used GeoTriples for generating and processing RML mappings over shapefiles, MonetDB and PostgreSQL.

[45] https://sentinel.esa.int/web/sentinel/missions/sentinel-2/

[46] https://sentinel.esa.int/web/sentinel/user-guides/sentinel-2-msi

[47] http://www.kadaster.nl/web/artikel/producten/TOP10NL.htm

[48] https://www.eea.europa.eu/data-and-maps/data/copernicus-land-monitoring-service-urban-atlas

7.4. Experimental Results

In Table 4 we present the elapsed time for generating the appropriate mappings for each dataset and subsequently processing them for generating an RDF graph. We made the generated datasets and the corresponding RML mappings publicly available as open data.[49,50,51]

As we report in Table 4, GeoTriples required 38 and 245 minutes respectively for generating and processing the relevant RML mappings for the Sentinel-2 and TOP10NL datasets.

For transforming the Urban Atlas dataset, we evaluated GeoTriples operating directly over shapefiles using the libraries GeoTools and GDAL, and GeoTriples operating over shapefiles stored in MonetDB and PostgreSQL. The elapsed times for each case are shown in Table 4. The resulting RDF graph has 106,020,936 triples but storage requirements differ for each case. All implementations store coordinates using the double-precision floating-point type, but choose to print different numbers of significant digits when serializing them as strings. The GDAL library uses 15 significant digits, GeoTools uses 9 significant digits, PostgreSQL uses 8 significant digits, and MonetDB uses 8 significant digits. As a result, the size of the output RDF graph in terms of disk usage differs. Small variations of the storage requirements also emerge from the different URI templates that are been used when operating directly on the files and when harvesting information from a relational database.

In Figure 5 we present the time required to generate RML mapping documents for the case of Urban Atlas assuming input shapefiles or spatially-enabled relational tables. We observe two clusters of measurements: one cluster for spatially-enabled relational databases and one for GeoTriples operating directly on shapefiles. The mapping generation process relies on accessing either the header of the shapefiles or the database catalogue which is measured to be a faster process. We notice that mapping generation is a very efficient process which is independent of the sizes of triples that will be generated eventually. Even in the worst case, it takes a few seconds. Comparing this with the elapsed times of Table 4, we can see that these times are essentially only for processing the mappings.

In Figure 6 we present the elapsed time for processing the generated RML mappings relatively to the number of generated triples. The variability of elapsed time

[49] http://datahub.io/dataset/sentinel2/

[50] http://datahub.io/dataset/top10nl/

[51] http://datahub.io/dataset/urban-atlas/

Dataset	# of files	Size on disk	# of subject maps	# of predicate object maps	# of triples	RDF graph size	Total conversion time
Sentinel-2, Level-1C products	2,188	186.4 MB	1,538	3,986	4,198,561	1.8 GB	38 min
TOP10NL	101	29 GB	29,135	190,898	480,346,004	99.0 GB	245 min
Urban Atlas	336	20 GB	672	5,042	106,020,936		
RML - Shapefiles - GDAL						51.8 GB	78 min
RML - Shapefiles - GeoTools						44.5 GB	72 min
RML - MonetDB						41.2 GB	77 min
RML - PostgreSQL						41.4 GB	64 min

Table 4: Datasets and first set of experiments

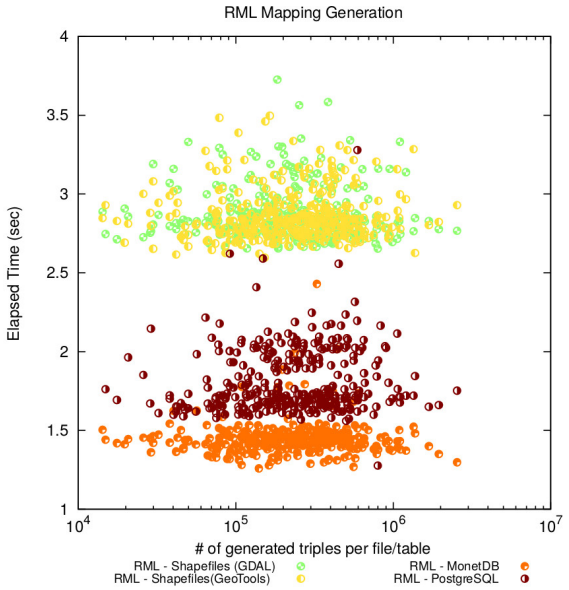


Figure 5: Generating RML mappings

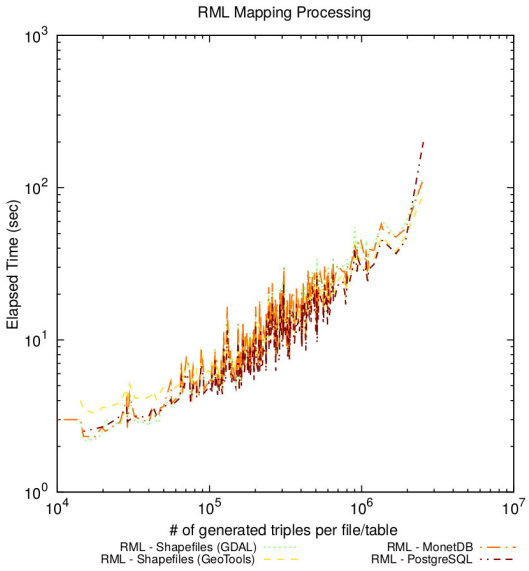


Figure 6: Processing RML mappings

for data points that correspond to the same number of triples comes from the fact that we may have two files with the same number of triples, but with very different sizes (e.g., 1GB vs. 2GBs) depending on the number of characters used to represent the geometries in the files. In Figure 7 we present the cumulative elapsed time for processing all RML mappings relatively to the total number of generated triples. Compared with Figure 6, the line now is smooth given that data points corresponding to the same number of triples are reported with one (cumulative) value. We observe that using PostgreSQL is the fastest approach followed by operating over shapefiles using GeoTools. Using GDAL or MonetDB is slower. In the case of MonetDB, the bottleneck is the time required to serialize the results and transport them to the client. For example, when serializing 99,064 geometries that exist in the `gr0011_athina` shapefile, 12 seconds are required for serializing all geometries while 37 seconds are required for transporting the data to the client.

7.5. The Experimental Setup for the Parallel Implementations

The second set of experiments were carried out in two different setups. One for the GeoTriples-Hadoop implementation and one for the other two implementations.

The Hadoop setup is composed by 16 virtual nodes, 1 for the master node and 15 for the cluster nodes. The master node has and Intel Xeon E5-4603 v2 CPU running at 2.20GHz with 2 cores. The system has 8GB of RAM, a 32GB virtual harddisk with EXT4 volume and an 82540EM Gigabit Ethernet Controller. The cluster

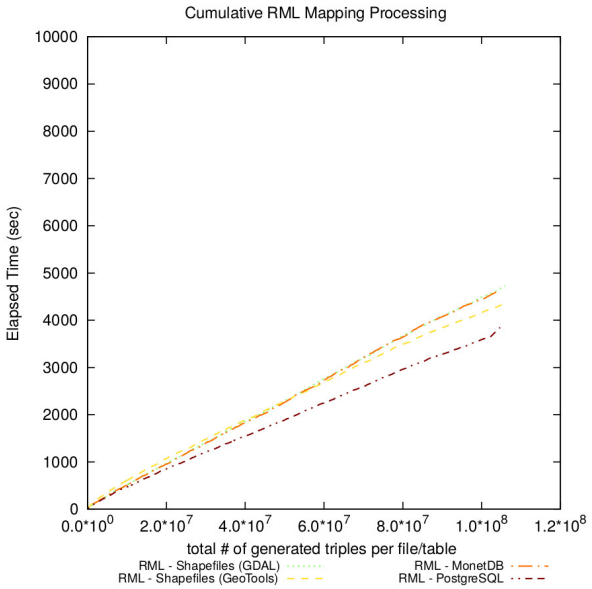


Figure 7: Processing RML mappings - cumulative

nodes belong to two groups in terms of computational power. 5 of them have a CPU with 2 cores, 4GB of RAM, a 32GB virtual harddisk with EXT4 volume and an 82540EM Gigabit Ethernet Controller. The rest 10 of them have a CPU with 2 cores, 4GB of RAM, a 32GB virtual harddisk with EXT4 volume and an NetXtreme II BCM5709 Gigabit Ethernet interface.

The experiments concerning GeoTriples and GeoTriples-Multi took place on a system with an Intel Xeon E5620 CPU running at 2.40GHz with 4 cores and each core has 8 threads. The system has 32GB of RAM (4 * DIMM DDR3 Synchronous 1333 MHz at 8GB) and a SCSI 1395GiB disk in RAID (MegaRAID SAS 2108).

7.6. Datasets for the Second Set of Experiments

The second set of experiments used three datasets composed of shapefiles and two datasets composed of CSV files. The shapefiles come from the GADM database of global administrative areas introduced in Section 5 and contain data for three different countries. The CSV files have been created from scratch for the purposes of our experiments.

The shapefile datasets are presented in Table 5. Each of the three datasets contains data for a different country available in the GADM portal. To create a variety of data sources, we have modified or copied the corresponding files many times. The first dataset contains 1000 copies of the shapefile for Ukraine. The file that contains the geometry is 2MB while the file that contains the non-spatial attributes is 20KB. With the Ukraine dataset, we wanted to check the performance of our implementations with many small files. The second data set is for Australia. The original file that contains the geometry is 20MB while the file that contains the non-spatial attributes is 8KB. We have modified these files by increasing the file that contains the geometry to 250MB, the file with the non-spatial data to 97KB, and by copying the file 60 times. With the Australia dataset, we wanted to check the efficiency of our implementation with a reasonable number of big files. The third data set is for Andorra. The original file that contains the geometry is 12KB while the file that contains the non-spatial attributes is 6KB. As with Australia, we have modified the Andorra dataset by increasing the file that contains the geometry to 625MB, the file with the non-spatial data to 264MB, and by copying the file 15 times. With the Andorra dataset, we wanted to check the efficiency of our implementation with few very big files.

Country	**.shp file size**	**.dbf file size**	**# of files**
Ukraine	2MB	20KB	1000
Australia	250MB	97KB	60
Andorra	625MB	264MB	15

Table 5: Shapefiles for the second set of experiments

Size	**# of records**	**# of files**
500MB	2.000.000	20
1GB	4.500.000	1

Table 6: CSV files for the second set of experiments

The CSV datasets are presented in Table 6. The first dataset is a CSV file with spatial and non-spatial data, its size is 500MB and it contains more than 2.000.000 records. We copy this file 20 times. The second dataset is a CSV file with spatial and non-spatial data, its size is 1GB and it contains more than 4.500.000 records.

The above shapefile and CSV datasets have been constructed so that appropriate features of the three implementations are tested in various interesting scenarios. This allowed us to make a more detailed comparison, something that we would have not achieved if we had used e.g., the whole Urban Atlas shapefile dataset we used in the first set of experiments.

7.7. Experimental Results

The results of our experiments for the three implementations are presented in Table 7. Before discussing the results in detail, we would like to mention that we

Dataset	Hadoop	Single	Multi
Shapefiles - Ukraine	16 min	69 min	16 min
Shapefiles - Australia	10 min	68 min	10 min
Shapefiles - Andorra	11 min	56 min	14 min
CSV files - first data set	18 min	133 min	20 min
CSV files - second data set	5 min	13 min	13 min

Table 7: Comparing the three GeoTriples implementations on shapefiles and CSV files

Shapefile	Distinct Records	Total Records
Ukraine	27 records	27 records
Australia	11 records	132 records
Andorra	7 records	367,388 records

Table 8: Number of records in each shapefile

have experimented with a variety of configurations of the GeoTriples-Hadoop and GeoTriples-Multi implementation. We performed experiments with different size of RAM and different block size (for Hadoop). Since we noticed only very minor fluctuations in the results, we will present results only for our default configuration of 4GB RAM and 128MB block size. We also performed experiments with having reducers that merge the output RDF files. Our experiments showed that the presence of reducers added 2-3 minutes to the elapsed time. Thus, we will not include reducers in the discussion below.

The first row of Table 7 contains the results for the Ukraine shapefile dataset. As shown in the table, the GeoTriples-Hadoop implementation allows for a decrease of 85% in the processing time compared with GeoTriples. GeoTriples-Hadoop and GeoTriples-Multi seem to have exactly the same performance. The reason for the latter result is that the files in the data set have small size and therefore the I/O intensity of the computation does not arise. A Hadoop implementation always suffers from a slow start given the time that Hadoop needs to communicates with the cluster and start the execution. In our implementation this time is 1-2 minutes. Moreover, the use of the Shapefile library adds additional time. We also noticed that the performance of GeoTriples for individual files is better than GeoTriples-Multi: GeoTriples needs around 4.15 seconds to process one file while GeoTriples-Multi needs 12.5 seconds. This occurs because of the competition for the common resources in GeoTriples-Multi and indicates that the performance will decrease as the parallel execution increases. GeoTriples-Hadoop needs 11-12 seconds to process one file independently of the number of cluster nodes, therefore, for large scale experiments GeoTriples-Hadoop is more efficient than GeoTriples-Multi.

The second row of Table 7 contains the results for the Australia shapefile dataset. As shown in the table, GeoTriples-Hadoop allows again for a 85% decrease in the processing time compared with GeoTriples. Since in the Australia dataset, individual files are larger than the ones in the Ukraine dataset, GeoTriples needs 68 seconds for processing a single file, GeoTriples-Multi needs 120-140 seconds and GeoTriples-Hadoop needs 110 seconds.

The third row of Table 7 contains the results for the Andorra shapefile dataset. As shown in the table, GeoTriples-Hadoop allows for an 80% decrease in the processing time compared with GeoTriples and a 21% decrease compared with GeoTriples-Multi. In other words, as the size of the data files increases, the difference in performance in favour of GeoTriples-Hadoop becomes clear. However, since we have significantly increased the size of each shapefile, the I/O and CPU intensity of the computation shows up for all implementations: GeoTriples needs 4 minutes for a single file, GeoTriples-Multi needs 14 minutes and GeoTriples-Hadoop needs 7 minutes. Overall the Hadoop implementation scales well given the parallel execution of tasks in the mappers.

The last two rows of Table 7 contain the results for the CSV datasets. For the first dataset, the GeoTriples-Hadoop allows for an 86% decrease in the processing time compared with GeoTriples and by a 10% decrease compared with GeoTriples-Multi. This increase in performance for GeoTriples-Hadoop is due mainly to the built-in mechanism of Apache Hadoop for the processing of CSV files. Apache Hadoop breaks down the input files in segments and balances the workload better than the GeoTriples or GeoTriples-Multi where load balancing does not take place. This fact is also proved by the last row of the table where we have a single CSV file and this will be processed in a single cycle by GeoTriples and GeoTriples-Multi while GeoTriples-Hadoop allows a decrease of 62% in the processing time compared with the other two implementations.

7.8. The Experimental Setup for the Comparison of GeoTriples and TripleGeo

The comparison of the two systems was carried out in the setup that was used for GeoTriples and GeoTriples-Multi (Section 7.5). For these experiments we set the minimum Java Heap Space memory to 2GB and maximum memory to 4GB.

Column	GeoTriples	TripleGeo Graph	TripleGeo RML	TripleGeo Stream
ID_0	X		X	
ISO	X		X	
NAME_0	X		X	
ID_1		X		X
NAME_1	X	X	X	X
HASC_1	X		X	
CCN_1	X		X	
CCA_1	X		X	
TYPE_1	X		X	
ENGTYPE_1	X	X	X	X
NL_NAME_1	X		X	
VARNAME_1	X		X	
Geometry	X	X	X	X

Table 9: The columns of every input record

Dataset	GeoTriples	TripleGeo Graph	TripleGeo RML	TripleGeo Stream
Shapefiles - Ukraine	109 min	82 min	67 min	71 min
Shapefiles - Australia	63 min	43 min	48 min	47 min
Shapefiles - Andorra	47 min	26 min	42 min	28 min

Table 10: Comparing GeoTriples and TripleGeo on shapefiles

RDF file	GeoTriples	TripleGeo Graph	TripleGeo RML	TripleGeo Stream
Ukraine	349 triples	216 triples	349 triples	216 triples
Australia	1,584 triples	88 triples	132 triples	1,056 triples
Andorra	4,408,656 triples	56 triples	30,912 triples	2,939,104 triples

Table 11: Number of triples in each RDF file

7.9. Datasets for the Third Set of Experiments

For this set of experiments we used the modified shapefiles that are presented in Table 5 and thoroughly described in Section 7.6.

7.10. Experimental Results

The results of the comparison are presented in Table 10. In order to have a fair comparison, we only measure the time that it takes GeoTriples to transform the input data given a mapping file (i.e., we do not measure the time it takes to generate the mappings).

Unlike GeoTriples, TripleGeo does not support transformation functions. In order to provide the systems with equivalent mapping files, the predicates that have to do with transformation functions were removed from the mapping file of GeoTriples.

The records of each shapefile given as input to the two systems have the same columns (Table 9). For every record of Andorra and Australia, the value of the attributes *CCA_1*, *NL_NAME_1* and *VARNAME_1* is *NULL*. Regarding the shapefile of Ukraine, the columns *CCA_1* and *NL_NAME_1* do not contain any information and moreover, two out of 27 records have a *NULL* value in the column *VARNAME_1*.

In Table 9 we can see that, GeoTriples and TripleGeo RML are able to transform 12 columns into RDF, whereas TripleGeo GRAPH and TripleGeo STREAM are able to transform only four attributes into RDF. GeoTriples and TripleGeo RML use the column *ID_1* as identifier and transform into RDF every other column of a record, unless its value is *NULL*. They also create three additional triples. The predicate *rdf:type* is

used in two of these triples to encode the class of each resource and the class of its geometry. The third triple has the predicate *geo:hasGeometry* to encode the geometry of a resource. For each record of Australia and Andorra, GeoTriples and TripleGeo RML will transform all of its columns, except from *CCA_1*, *NL_NAME_1* and *VARNAME_1* whose values are *NULL*, into RDF. If we also take into consideration Table 9 and the additional triples, we can see that both tools create 12 triples for each record of Australia and Andorra. In the same manner, for 25 out of the 27 records of Ukraine, the tools create 13 RDF triples and for the other two records, whose value in the column *VARNAME_1* is *NULL*, they create 12 RDF triples. Every RDF file that is produced by GRAPH and STREAM modes of TripleGeo is also enriched with three triples that contain the *rdf:type* predicate and one triple that contains the *geo:hasGeometry* predicate. We see that these modes have one additional triple that contains the *rdf:type* predicate than GeoTriples and TripleGeo RML. This additional triple encodes the fact that each resource is an instance of *ogc:Feature*. As a result, TripleGeo GRAPH produces eight triples for each distinct record, whereas TripleGeo STREAM produces eight triples for every record. The GRAPH and STREAM modes of TripleGeo transform into RDF the columns that are marked in Table 9. Tables 8 and 11 provide information about the input and output files.

The experimental results (Table 10) show that TripleGeo outperforms GeoTriples. We expected that the STREAM and GRAPH modes of TripleGeo would need less time to produce the RDF files, since TripleGeo transforms fewer attributes in these modes than GeoTriples hence it needs less I/O operations. The results also show that the RML mode of TripleGeo is performing better than GeoTriples, when it comes to small input files (i.e., Ukraine). The performance gap is smaller when larger files (i.e., Australia and Andorra) are given as input to the systems, even though GeoTriples carries out more I/O operations, since it generates significantly more triples than the RML mode of TripleGeo. TripleGeo RML produces fewer triples than GeoTriples because it removes some of the duplicate triples that will be generated given the artificial way of increasing the sizes of the datasets by duplicating the records of the shapefiles of Australia and Andorra (see Section 7.6).

Finally, the results that have been presented in Section 7.7, show that GeoTriples-Hadoop and GeoTriple-Multi not only outperform the simple implementation of GeoTriples, but also every mode of TripleGeo.

8. Summary and Conclusions

We presented the tool GeoTriples which is able to transform geospatial data stored in raw files and spatially-enabled RDBMS to RDF graphs using well-known vocabularies. The tool works in three steps. First, it generates automatically extended R2RML or RML mappings that can be used to transform the input data into RDF. As an optional second step, the user may revise these mappings according to her needs e.g., to utilize a different vocabulary. Finally, GeoTriples processes these mappings and produces an RDF graph. We performed a detailed performance evaluation of three implementations of GeoTriples and showed that all of them can deal with large data volumes (in the order of Gigabytes). We also compared GeoTriples with the similar tool TripleGeo and found that TripleGeo is more efficient than the single node implementation of GeoTriples. Finally, a Hadoop-based implementation of GeoTriples outperforms all other implementations of GeoTriples or TripleGeo.

In this paper we have seen that GeoTriples and TripleGeo are both state-of-the-art tools for transforming geospatial data into RDF. However, transforming geospatial data from their native formats into RDF is only one phase of the life-cycle of linked geospatial data which we have presented in [22]. The requirements of this life-cycle are addressed by the linked geospatial data tool suite that has been developed by our group at the National and Kapodistrian University of Athens and has been recently surveyed in [20]. All our tools are open source and they will continue to be maintained by our team while they are being used in research projects by our group or others. Motivated by the good performance of GeoTriples-Hadoop, in future work, we will be concentrating on making all of our tools scale to even bigger datasets by utilizing big data technologies like HopsFS [25] and Spark [33].

Acknowledgments

This work has been funded in part by the European FP7 project LEO (611141), the FP7 project MELODIES (603525), the Dutch NWO project COMMIT, and the H2020 project Copernicus App Lab (730124).

References

[1] Open Geospatial Consortium. OpenGIS Implementation Standard for Geographic information - Simple feature access - Part 1: Common Architecture. OpenGIS Implementation Standard,

2010. Available from: http://portal.opengeospatial.org/files/?artifact_id=25355.

[2] Open Geospatial Consortium. GeoSPARQL - A geographic query language for RDF data. OpenGIS Implementation Standard, November 2012. Available from: https://portal.opengeospatial.org/files/?artifact_id=47664.

[3] S. Auer, J. Lehmann, and S. Hellmann. LinkedGeoData: Adding a Spatial Dimension to the Web of Data. In A. Bernstein, D. Karger, T. Heath, L. Feigenbaum, D. Maynard, E. Motta, and K. Thirunarayan, editors, *The Semantic Web - ISWC 2009*, volume 5823 of *Lecture Notes in Computer Science*, pages 731–746. Springer Berlin Heidelberg, 2009.

[4] K. Bereta and M. Koubarakis. Ontop of geospatial databases. In *The Semantic Web - ISWC 2016 - 15th International Semantic Web Conference, Kobe, Japan, October 17-21, 2016, Proceedings, Part I*, pages 37–52, 2016.

[5] K. Bereta and M. Koubarakis. Creating Virtual Semantic Graphs on Top of Big Data from Space. In *Proc. of the 2017 Conference on Big Data from Space (BiDS)*, 2017.

[6] K. Bereta, P. Smeros, and M. Koubarakis. Representation and querying of valid time of triples in linked geospatial data. In P. Cimiano, O. Corcho, V. Presutti, L. Hollink, and S. Rudolph, editors, *The Semantic Web: Semantics and Big Data*, volume 7882 of *Lecture Notes in Computer Science*, pages 259–274. Springer Berlin Heidelberg, 2013.

[7] K. Bereta, G. Xiao, M. Koubarakis, M. Hodrius, C. Bielski, , and G. Zeug. Ontop-spatial: Geospatial data integration using GeoSPARQL-to-SQL translation. In *Proceedings of the 15th International Semantic Web Conference, Kobe, Japan*, 2016. Demo paper.

[8] T. Berners-Lee. Relational Databases on the Semantic Web. http://www.w3.org/DesignIssues/RDB-RDF.html.

[9] A. Bertails, M. Arenas, E. Prudhommeaux, and J. Sequeda. A Direct Mapping of Relational Data to RDF. W3C Recommendation, 2012. Available from: http://www.w3.org/TR/rdb-direct-mapping/.

[10] S. Brüggemann, K. Bereta, G. Xiao, and M. Koubarakis. Ontology-based data access for maritime security. In *The Semantic Web. Latest Advances and New Domains - 13th International Conference, ESWC 2016, Heraklion, Crete, Greece, May 29 - June 2, 2016, Proceedings*, pages 741–757, 2016.

[11] K. Chentout and A. A. Vaisman. Adding Spatial Support to R2RML Mappings. In *OTM Workshops*, volume 8186 of *Lecture Notes in Computer Science*. Springer, 2013.

[12] Y. Chronis, Y. Foufoulas, V. Nikolopoulos, A. Papadopoulos, L. Stamatogiannakis, C. Svingos, and Y. E. Ioannidis. A relational approach to complex dataflows. In *Proceedings of the Workshops of the EDBT/ICDT 2016 Joint Conference, EDBT/ICDT Workshops 2016, Bordeaux, France, March 15, 2016.*, 2016.

[13] R. Cyganiak, C. Bizer, J. Garbers, O. Maresch, and C. Becker. The D2RQ Platform. Available from: http://d2rq.org/.

[14] S. Das, S. Sundara, and R. Cyganiak. R2RML: RDB to RDF Mapping Language. W3C Recommendation, 2012. Available from: http://www.w3.org/TR/r2rml/.

[15] A. de León, V. Saquicela, L. M. Vilches, B. Villazón-Terrazas, F. Priyatna, and O. Corcho. Geographical linked data: A Spanish use case. In *Proceedings of the 6th International Conference on Semantic Systems*, I-SEMANTICS '10, pages 36:1–36:3, New York, NY, USA, 2010. ACM.

[16] J. Dean and S. Ghemawat. Mapreduce: a flexible data processing tool. *Communications of ACM*, 53(1):72–77, 2010.

[17] A. Dimou, D. Kontokostas, M. Freudenberg, R. Verborgh, J. Lehmann, E. Mannens, S. Hellmann, and R. V. de Walle. Assessing and Refining Mappings to RDF to Improve Dataset Quality. In *The Semantic Web - ISWC 2015 - 14th International Semantic Web Conference, Bethlehem, PA, USA, October 11-15, 2015, Proceedings, Part II*, pages 133–149, 2015.

[18] A. Dimou, M. Vander Sande, P. Colpaert, R. Verborgh, E. Mannens, and R. Van de Walle. RML: A generic language for integrated RDF mappings of heterogeneous data. In *Proceedings of the 7th Workshop on Linked Data on the Web*, April 2014.

[19] M. J. Egenhofer and R. D. Franzosa. Point-Set Topological Spatial Relations. In *Proceedings of International Journal of Geographical Information Systems*, volume 5(2), pages 161–174, 1991.

[20] M. Koubarakis, K. Bereta, G. Papadakis, D. Savva, and G. Stamoulis. Big, linked geospatial data and its applications in Earth observation. *IEEE Internet Computing*, 21(4):87–91, 2017.

[21] M. Koubarakis, M. Karpathiotakis, K. Kyzirakos, C. Nikolaou, and M. Sioutis. Data models and query languages for linked geospatial data. In *Invited tutorial at the 8th Reasoning Web Summer School 2012 (RW 2012). In: Eiter, T., Krennwallner, T. (eds.) Reasoning Web. Semantic Technologies for Advanced Query Answering. Lecture Notes in Computer Science, vol. 7487, pp. 290-328. Springer.*, 2012.

[22] M. Koubarakis, K. Kyzirakos, C. Nikolaou, G. Garbis, K. Bereta, R. Dogani, S. Giannakopoulou, P. Smeros, D. Savva, G. Stamoulis, G. Vlachopoulos, S. Manegold, C. Kontoes, T. Herekakis, I. Papoutsis, and D. Michail. Managing big, linked, and open Earth observation data: Using the teleios/leo software stack. *IEEE Geoscience and Remote Sensing Magazine*, 4(3):23–37, Sept 2016.

[23] K. Kyzirakos, M. Karpathiotakis, G. Garbis, C. Nikolaou, K. Bereta, I. Papoutsis, T. Herekakis, D. Michail, M. Koubarakis, and C. Kontoes. Wildfire monitoring using satellite images, ontologies and linked geospatial data. *Journal of Web Semantics*, 24:18–26, 2014.

[24] K. Kyzirakos, M. Karpathiotakis, and M. Koubarakis. Strabon: A semantic geospatial DBMS. In P. Cudré-Mauroux, J. Heflin, E. Sirin, T. Tudorache, J. Euzenat, M. Hauswirth, J. X. Parreira, J. Hendler, G. Schreiber, A. Bernstein, and E. Blomqvist, editors, *International Semantic Web Conference (1)*, volume 7649 of *Lecture Notes in Computer Science*, pages 295–311. Springer, 2012.

[25] S. Niazi, M. Ismail, S. Haridi, J. Dowling, S. Grohsschmiedt, and M. Ronström. Hopsfs: Scaling hierarchical file system metadata using newsql databases. In *15th USENIX Conference on File and Storage Technologies, FAST 2017, Santa Clara, CA, USA, February 27 - March 2, 2017*, pages 89–104, 2017.

[26] K. Patroumpas, M. Alexakis, G. Giannopoulos, and S. Athanasiou. TripleGeo: an ETL Tool for Transforming Geospatial Data into RDF Triples. In *Proceedings of the Workshops of the EDBT/ICDT 2014 Joint Conference (EDBT/ICDT 2014)*, volume 1133 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2014.

[27] D. A. Randell, Z. Cui, and A. G. Cohn. A Spatial Logic based on Regions and Connection. In *Proceedings of the 3rd International Conference on Principles of Knowledge Representation and Reasoning (KR'92). Cambridge, MA, October 25-29, 1992.*, pages 165–176, 1992.

[28] M. Rodriguez-Muro and M. Rezk. Efficient SPARQL-to-SQL with R2RML mappings. *J. Web Sem.*, 33:141–169, 2015.

[29] M. A. Sherif, K. Dreßler, P. Smeros, and A. Ngonga Ngomo. Radon - rapid discovery of topological relations. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA.*, pages 175–181, 2017.

[30] P. Smeros and M. Koubarakis. Discovering spatial and temporal links among RDF data. In *Proceedings of the Workshop on*

Linked Data on the Web, LDOW 2016, co-located with 25th International World Wide Web Conference, 2016.

[31] C. Stadler, J. Lehmann, K. Höffner, and S. Auer. Linkedgeodata: A core for a web of spatial open data. *Semantic Web*, 3(4):333–354, 2012.

[32] D. Steer. SquirrelRDF, 2006. Available from: `https://sourceforge.net/projects/jena/files/Archive/SquirrelRDF/`.

[33] M. Zaharia, R. S. Xin, P. Wendell, T. Das, M. Armbrust, A. Dave, X. Meng, J. Rosen, S. Venkataraman, M. J. Franklin, A. Ghodsi, J. Gonzalez, S. Shenker, and I. Stoica. Apache Spark: a unified engine for big data processing. *Commun. ACM*, 59(11):56–65, 2016.