NATIONAL AND KAPODISTRIAN UNIVERSITY OF ATHENS

SCHOOL OF SCIENCE

DEPARTMENT OF INFORMATICS AND TELECOMMUNICATIONS

POSTGRADUATE STUDIES

MASTER THESIS

GeoTriples: A Tool For Publishing Earth Observation and Geospatial Data as RDF Graphs Using the R2RML Mapping Language

Ioannis Vlachopoulos

Supervisors: **Manolis Koubarakis**, Professor UoA
Kostis Kyzirakos, Post-Doctoral Researcher CWI

ATHENS

JANUARY 2015

ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΜΕΤΑΠΤΥΧΙΑΚΕΣ ΣΠΟΥΔΕΣ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

GeoTriples: Εφαρμογή για Δημοσιοποίηση Δεδομένων Παρατήρησης Γης και Γεωχωρικών Δεδομένων σε Μορφή RDF Γράφων, Χρησιμοποιώντας τη Γλώσσα Αντιστοίχησης R2RML

Ιωάννης Βλαχόπουλος

Επιβλέποντες: **Μανόλης Κουμπαράκης**, Καθηγητής ΕΚΠΑ
Κωστής Κυζηράκος, Μεταδιδακτορικός Ερευνητής CWI

ΑΘΗΝΑ

ΙΑΝΟΥΑΡΙΟΣ 2015

MASTER THESIS

GeoTriples: A Tool For Publishing Earth Observation and Geospatial Data as RDF Graphs Using the R2RML Mapping Language

Ioannis Vlachopoulos

R.N.: M1249

SUPERVISORS:

Manolis Koubarakis, Professor UoA

Kostis Kyzirakos, Post-Doctoral Researcher CWI

EXAMINATION COMMITEE:

Efstathios Hadjiefthymiades, Assistant Professor UoA

ATHENS

JANUARY 2015

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

GeoTriples: Εφαρμογή για Δημοσιοποίηση Δεδομένων Παρατήρησης Γης και Γεωχωρικών Δεδομένων σε Μορφή RDF Γράφων, Χρησιμοποιώντας τη Γλώσσα Αντιστοίχησης R2RML

Ιωάννης Βλαχόπουλος

Α.Μ.: M1249

ΕΠΙΒΛΕΠΟΝΤΕΣ:

Μανόλης Κουμπαράκης, Καθηγητής ΕΚΠΑ

Κωστής Κυζηράκος, Μεταδιδακτορικός Ερευνητής CWI

ΕΞΕΤΑΣΤΙΚΗ ΕΠΙΤΡΟΠΗ:

Ευστάθιος Χατζηευθυμιάδης, Αναπληρωτής Καθηγητής ΕΚΠΑ

ΑΘΗΝΑ

ΙΑΝΟΥΑΡΙΟΣ 2015

Abstract

A plethora of Earth Observation data that is becoming available at no charge in Europe and the US recently reflects the strong push for more open Earth Observation data. Linked Data is a paradigm which studies how one can make data available on the Web and interconnect it with other data with the aim of making the value of the resulting "Web of data" greater than the sum of its parts. Open Earth Observation data that are currently made available by space agencies such as ESA and NASA are not following the linked data paradigm. Therefore, Earth Observation data and other kinds of geospatial data that are necessary for a user to satisfy her information needs can only be found in different data silos, where each silo may contain only part of the needed data. Publishing the content of these silos as RDF graphs, enables the development of data analytics applications with great environmental and financial value. In this thesis, we present the tool GeoTriples that allows for the transformation of Earth Observation data and geospatial data into RDF graphs. GeoTriples goes beyond the state of the art by extending the R2RML mapping language to be able to deal with the specificities of geospatial data. GeoTriples is a semi-automated tool that allows the publication of geospatial information into an RDF graph using the state of the art vocabularies like GeoSPARQL and stSPARQL, but at the same time it is not tightly coupled to a specific vocabulary.

SUBJECT AREA: Semantic Web

KEYWORDS: linked data, earth observation, geospatial data, RDF graphs, mapping, R2RML

Περίληψη

Τα τελευταία χρόνια ένας ολοένα αυξανόμενος όγκος δεδομένων παρατήρησης γης γίνεται διαθέσιμος στην Ευρώπη και την Αμερική. Τα συνδεδεμένα δεδομένα είναι ένα μοντέλο το οποίο μελετάει τον τρόπο με τον οποίο τα δεδομένα μπορούν να γίνουν διαθέσιμα στον παγκόσμιο ιστό και να διασυνδεθούν με άλλα δεδομένα, δημιουργώντας επομένως έναν Ïστό Δεδομένων¨. Ωστόσο τα δεδομένα παρατήρησης γης που διατίθενται από υπηρεσίες όπως η ESA και η NASA δεν ακολουθούν το μοντέλο των συνδεδεμένων δεδομένων. Κατά συνέπεια, προκειμένου κάποιος χρήστης κάποιος χρήστης να ικανοποιήσει διαφόρου τύπου ανάγκες για πληροφορίες, θα πρέπει να συλλέξει γεωχωρικά δεδομένα και δεδομένα παρατήρησης γης από διαφορετικά σιλό. Δημοσιεύοντας τα δεδομένα των σιλό αυτών ως γράφους RDF, καθίσταται δυνατή η ανάπτυξη εφαρμογών ανάλυσης δεδομένων με μεγάλη περιβαλλοντολογική και οικονομική αξία. Στην παρούσα διπλωματική, παρουσιάζεται το εργαλείο GeoTriples για το μετασχηματισμό δεδομένων παρατήρησης γης και γεωχωρικών δεδομένων σε γράφους RDF. Το GeoTriples επεκτείνει τη γλώσσα αντιστοίχησης R2RML ώστε να λάβει υπόψιν και τις ιδιαιτερότητες που παρουσιάζουν τα γεωχωρικά δεδομένα. Αποτελεί μία ημι-αυτόματη εφαρμογή για μετατροπή γεωχωρικής πληροφορίας σε RDF χρησιμοποιώντας δημοφιλή λεξιλόγια όπως GeoSPARQL και stSPARQL, χωρίς ταυτόχρονα να δεσμεύεται αποκλειστικά με κάποιο από αυτά.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: Σημασιολογικός Ιστός

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ : Συνδεδεμένα δεδομένα, παρατήρηση γης, γεωχωρικά δεδομένα, γράφοι RDF, αντιστοίχηση, R2RML

Ευχαριστίες

Αρχικά θα ήθελα να ευχαριστήσω τον καθηγητή και επιβλέποντα αυτής της διπλωματικής εργασίας, Μανόλη Κουμπαράκη, για την πολύτιμη καθοδήγησή του η οποία κατέστησε δυνατή την εκπόνησή της εργασίας αυτής. Επίσης θα ήθελα να ευχαριστήσω το μεταδιδακτορικό ερευνητή Κωστή Κυζηράκο και το συνάδελφο Δημητριανό Σάββα για τη συνεργασία τους και την πολύτιμη βοήθειά τους στο ερευνητικό έργο LEO, στα πλαίσια του οποίου εκπονήθηκε η εργασία.

Contents

List of Figures

List of Tables

Chapter 1
Introduction

Lots of Earth Observation (EO) data has become available at no charge in Europe and the US recently and there is a strong push for *more open EO data*. For example, a recent paper on Landsat data use and charges by the US National Geospatial Advisory Committee - Landsat Advisory Group starts with the following overarching recommendation: "Landsat data must continue to be distributed at no cost". Similarly, the five ESA Sentinel satellites that would soon go into orbit have already adopted a fully open and free data access policy. In the same spirit, the Autumn 2012 edition of the newsletter of the European Association of Remote Sensing Companies makes the case for a free and open data policy for GMES, arguing that this will fuel the development of the geo-information services industry, and as a result, bring maximum economic benefit to Europe.

Linked data is a new data paradigm which studies how one can make RDF data available on the Web, and interconnect it with other data with the aim of increasing its value [9]. In the last few years, linked *geospatial* data has received attention as researchers and practitioners have started tapping the wealth of geospatial information available on the Web [21]. As a result, the *linked open data (LOD) cloud* has been rapidly populated with geospatial data some of it describing EO products (e.g., CORINE Land Cover and Urban Atlas published by project TELEIOS[1]). The abundance of this data can prove useful to the new missions (e.g., Sentinels) as a means to increase the usability of the millions of images and EO products that are expected to be produced by these missions. However, open EO data that are currently made available by space agencies such as ESA and NASA are *not* following the linked data paradigm. Therefore, from the perspective of a user, the EO data and other kinds of geospatial data necessary to satisfy his or her information need can only be found in different data silos, where each silo may contain only part of the needed data. *Opening up these silos* by publishing their contents as RDF and interlinking them with semantic connections will allow the development of data analytics applications with great environmental and financial value.

In this dissertation we will describe GeoTriples, a tool for translating EO and geospatial data into RDF graphs, using the R2RML mapping language. The tool is developed in the

[1] `http://www.earthobservatory.eu`

context of the FP7 project LEO[2]. LEO aims to study the life cycle of linked open EO data and develop tools that support it [20]. More specifically, the scientific and technical objectives of LEO are the following:

- To capture, as precisely as possible, the life cycle of linked open EO data.
- To develop publishing tools that transform open EO data and metadata, made available by space agencies such as ESA and NASA, from their standard formats into RDF and make it available on the LOD cloud.
- To develop publishing tools that transform open geospatial data and metadata from their standard formats into RDF and make it available on the LOD cloud. Open geospatial data (e.g., digital maps, administrative data, enviromental data, etc.) are typically used together with EO data in applications such as precision farming and are made available by public agencies as well (e.g., the Bavarian Topological Survey for the precision farming application developed in LEO. This is also the main objective of this dissertation.
- To develop tools that interlink open EO data sources and geospatial data sources published as RDF.
- To develop tools for cross-platform searching, browsing and visualization of linked EO data and linked geospatial data.
- To demonstratate the value of the developed tools by:
  - Performing large-scale publication and linking of open EO data from the GMES Space Component Data Access warehouse managed by ESA, and relevant geospatial datasets made available by other public bodies in Europe.
  - Developing a precision farming application that shows how geo-information services based on linked EO data, linked geospatial data and specialized algorithms can contribute to an environmental friendly increase in the efficiency of agricultural production.

The rest of this disseration is as follows; First of all, we will report work carried out in mapping techniques from the relational model to the RDF model. Then, we will describe the various formats of EO and geospatial data made publicly available and report on related work on publishing such data as RDF graphs. Finally, we are going to present GeoTriples,

[2] `http://www.linkedeodata.eu`

developed in the context of this thesis and project LEO, and examples of its usage in a real world scenario.

Chapter 2
Languages for mapping relational databases to RDF graphs

In the current state of the Web, the majority of data is stored in relational databases (RDB). Therefore, there is a need to bridge the conceptual gap between the standard relational model and the Resource Description Framework (RDF) that is the de facto standard for publishing and linking semantic data on the Web. Depending on current requirements that need to be addressed, different mapping methods may need to be employed. In this chapter, we describe the state of the art RDB-to-RDF mapping languages designed for this task. For completeness, we will not only present mapping languages, but also present existing systems and prototypes that perform such a mapping without the formal definition of a mapping language. Then we will report a feature based comparison that categorizes current approaches in four categories: the Direct Mapping approach, the read-only general-purpose mapping approach, the read-write general-purpose approach and the special-purpose approach. Then, we will present in more detail the R2RML mapping language, that recently became a W3C recommendation, for expressing such mappings and we will present systems that process R2RML mappings.

2.1 Mapping languages and systems

We start our discussion of mapping languages by presenting the **Direct Mapping** of Relational Data to RDF[1] approach that became a W3C recommendation in 2012. The direct mapping approach is a straight forward method for translating relational data into RDF. According to this approach, relational tables are mapped to classes defined by an RDF vocabulary, while the attributes of each table are mapped to RDF properties that represents the relation between subject and object resources. Identifiers, classes, properties, and instances are generated automatically. According to this approach, the generation of RDF data is dictated by the schema followed by the relational database. The directed mapping mechanism was initially defined in [7] while [1] is an implementation of such a mechanism.

[1] `http://www.w3.org/TR/2012/REC-rdb-direct-mapping-20120927/`

A lightweight tool designed to publish relational data in the Semantic Web is **Triplify**. Its main functionality lies on mapping HTTP-URI requests onto relational database queries, and translating the results into RDF statements. The rationale behind this functionality is that the largest percentage of data is already stored in structured form but are published as HTML by web applications. The main objectives of Triplify are the following; a) to enable users to publish several formats of data as linked data, e.g., JSON and CSV from existing web applications, b) offer preconfigured mappings as input to popular content management systems such as Drupal and Wordpress, c) perform selectively updates on large amount of data without having to re-crawl unchanged content and d) evaluate the flexibility and scalability of its approach using the data published by the Open Street Map project. Triplify needs three configuration elemenents to generate RDF triples. These elements are; a default schema namespace, a mapping from namespace prefixes to namespace URIs and a mapping of URL patterns to sets of SQL queries. The main notion on which Triplify is based, is the fact that for each logical table (a view or a physical table), the unique identifier is used for the production of the instance URIs. The names of the columns are used to generate object and data properties, and, finally, the values of the SQL result set are used to construct the objects of the RDF triples, which can be literal values or references to other objects. It is, however, important to do some preprocessing via SQL to generate these views and convert them into the RDF data model. This preprocessing involves the use of several extensions of SQL that annotate inline generated the views generated and remain transparent to the SQL processor. The extensions used are the *mapping to existing vocabularies* (e.g., column name "name" to "foaf:name"), object properties in order to make reference to other URIs as automatic mechanisms consider all columns by default to be datatype properties, datatypes and language tags to properly annotate RDF literals.

Another popular mapping language to translate relational data into RDF, is **D2RQ** [13, 10]. Being the successor of the XML-based D2R MAP [8], it treats non-RDF relational databases as virtual RDF graphs. It exposes relational data as RDF triples and allows the access to it via SPARQL queries. Mappings can be produced either automatically or given as input by the user, in case the re-use of an existing vocabulary is needed. The mappings are expressed as RDF triples using the TURTLE syntax. A D2RQ mapping is based on the notion of the ClassMap. The ClassMap represents a class or a group of classes in an ontology. It specifies whether URI instances derive from blank nodes or by the combination of a URI pattern with a primary key value. Each ClassMap, in turn, consists of a set of property bridges which specify how properties on the instance level are created and how database values are mapped to URIs or literals. There are two types of property bridges: the datatype property bridges whose values are actually the ones stored in the relational

database as they are, and the object property bridges that are used for object properties in order to make references to other URI instances.

In order to standardize RDB-to-RDF mappings, there has been an ongoing Working Group accommodated by the W3C RDB2RDF. The result of this work is the design of the language **R2RML** which provides features similar to D2RQ except for having some extra ones like Named Graph and SQL Views support [14]. More details about R2RML are given in Section 2.3.

R_2O [6] is a RDB-to-ontology mapping language, fully declarative and extensible. The ontologies may be expressed either in RDFS or OWL. The mappings are expressed in a XML-based syntax. An R_2O mapping defines how to create instances in the ontology and can be used to populate the ontology automatically using the data extracted from the relational database. Such a mapping can also be processed by other tools, middleware or APIs for tasks such as self-verification in order to automatically detect inconsistencies on the mapping itself, to verify the integrity of parts of a DB by applying ontology axioms to the DB elements and automatically characterize data sources to allow dynamic query distribution in intelligent information integration approaches. However, R_2O mappings are not bidirectional and thus, only RDB2RDF mapping is supported. Additionally, the mappings are not intended to be human-readable and as a result are not to be generated or edited manually, except via a graphical user interface. The ODEMapster[2] is a tool that has been enhanced to process R_2O mappings.

[15] introduces an OWL-based representation format for relational data and schema component, the **Relational.OWL**. It defines an OWL ontology to describe the RDB schema and its corresponding data. This mapping is targeted to be used in data-exchange in peer-to-peer databases. Each peer must be able to process and understand OWL documents, have access to the subject Relational.OWL ontology or a semantically equivalent one and also have access to the OWL schema representation of the corresponding database. As it is OWL-based, one of its main aspects is the ability to represent knowledge, since, applied to the domain of relational databases, data and schema items can be described in a machine readable but also understandable format, provided that an ontology is specified for the relational data. In addition, reliability is ensured for data and schema exchange between peers of a network. However, in order to minimize the risk of misunderstanding the knowledge represented, it is important to define a representation format which is unambiguous for all the peers involved, even though it may not be stated explicitly. A Relational.OWL mapping also contains selected metadata of the target database. This metadata mostly includes

[2] `http://neon-toolkit.org/wiki/ODEMapster`

information about tables and columns, primary and foreign key constraints and datatypes.

Openlink Software developed the **Virtuoso Universal Server** which, via RDF views, exposes relational data on the Semantic Web [16, 27].The Virtuoso RDF views are designed to translate SQL query result sets into RDF triples using mappings expressed in a declarative Meta Schema Language in combination with pre-existing vocabularies. The Meta Schema Language resembles to some extent SQL DDL from a syntax point of view.

R3M [17, 18] is another RDB to RDF mapping language that records additional information about the database schema to support update operations. An R3M mapping is expressed in RDF, using the R3M ontology, and maps the terms of a domain ontology model to the database schema. Its main approach is similar to other existing mapping languages; database tables are mapped to ontology classes and table columns are mapped to data properties. R3M also supports mapping link tables to object properties instead of classes, in order to define relations between URI instances. The root element of a R3M mapping is called the DatabaseMap. It represents the database and access information for the mediator. The URI prefix of the RDF dataset to be produced is also provided in the DatabaseMap along with a list of target tables to be mapped which are called TableMaps. A TableMap represents the mapping of an individual table. It provides the ontology class in which the instances are assigned, as well as a URI pattern for them. It also contains a list of AttributeMaps which maps the table's columns to properties defined in the ontology. Finally, R3M mappings support also insert/update/delete operations on the target relational database. These mappings instruct how SPARQL update operations such as INSERT DATA, DELETE DATA and MODIFY can be translated into SQL.

2.2 Comparison of mapping languages

A comparison framework is defined in [19] for the functional juxtaposition of the mapping languages presented in this chapter. Since all languages support the direct mapping approach, several features that are expected from a mapping language are defined. The comparison framework of [19] comprises the following features.

1. **Logical Table to Class**: A logical table represents a relational table that may not be materialized within the RDBMS. Thus, a logical table can be an SQL view or an ad-hoc SQL query. This feature maps a logical table to an RDFS class.

2. **M:N Relationships**: Relational databases use special tables, called cross-reference tables, for representing M:N relationships among entities. This feature maps such ta-

bles to the appropriate RDF constructs, since such relationships are expressed naturally in RDF using multiple triples.

3. **Project Attributes**: An RDB to RDF mapping may choose to export into RDF a subset of all available columns within a table. For example, sensitive data like personal, private, confidential or classified information should not be exported from the database. Therefore, the mappings should project only the required attributes in the output RDF graph.

4. **Select Conditions**: There are occasions in which not all the records should be mapped to triples, but only those satisfying a user-defined condition. A mapping language should allow the definition of such a selection predicate that would allow only qualifying records to be exported into RDF.

5. **User-defined Instance URI**: Apart for sequences of URI that are automatically generated by the mapping engine, the mapping language should provide a templating mechanism for the generated URI so that the user could customize the form of the produced URI.

6. **Literal to URI**: The SQL standard does not provide a specialized datatype for representing a URI. In practice, a URI is represented as a set of characters. Therefore, such a value should be converted to a valid URI in the RDF representation.

7. **Vocabulary Reuse**: The target vocabulary for a mapping can either be derived automatically by taking into account table and attributes names, or could be mapped to an existing vocabulary. This feature allows the mapping to use terms from an existing vocabulary.

8. **Transformation Functions**: Values of a relational table may need an intermediate transformation step prior exporting them into RDF. This feature allows the expression of transformation functions on relation data for converting them into the desired form that will be exported into RDF.

9. **Datatypes**: Although RDF uses XML datatypes, and a standardized mapping between SQL datatypes and XML datatypes exists[3], a mapping language should allow the assertion of the datatype information. This feature allows the explicit definition of the datatype of the target RDF literal within a mapping.

[3]ISO/IEC 9075-14 XML-related specifications, `http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=53686`

Table 2.1: Comparison of RDB-to-RDF mapping languages[19]

	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11	F12	F13	F14	F15
Direct Mapping	(✓)	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓
R2O	✓	✗	✓	✓	✓	✓	✓	✓	✓	✗	✓	(✓)	✗	(✓)	✗
Relatinal OWL	(✓)	✗	✓	✗	✗	✗	✗	✗	✓	✗	✓	(✓)	✗	✗	✓
Virtuoso	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	(✓)	✗	✓	✗
D2RQ	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗	✓	(✓)	✓	✓	✗
Triplify	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗	✗	(✓)	✗	✓	✗
R2RML	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	(✓)	✓	✓	✗
R3M	(✓)	✓	✓	(✓)	✓	✓	✓	✓	✓	✗	✗	✓	✗	✓	✓

✓= full support (✓) = partial support ✗= no support

10. **Named Graphs**: An RDF dataset may contain multiple graphs. A mapping language should allow the assignment of specific fragments of the database to different named graphs instead of being forced to assign all produced triples into a single graph.

11. **Blank Nodes**: Blank nodes are RDF terms that denote anonymous identifiers that are unique within an RDF graph, but their value is of no importance. Blank nodes are being used by this feature to map tuples that do not have a unique identifier such as a primary key.

12. **Integrity Constraints**: An important functionality of a relational database is to allow the specification of integrity constraints and to enforce them. The authors of [19] distinguish between key constraints (primary key, foreign key) and other constraints (not null, unique, check). This feature allows the explicit declaration of such constraints in the mapping language.

13. **Static Metadata**: The mapping language should allow the addition of static metadata into the produced RDF graph. For example, the user should be able to provide information (e.g., provenance information) that is not explicitly present in the relational database.

14. **One table to n Classes**: In most cases, the RDF resources generated from a logical table are instances of a single RDFS class. This feature allows us to map one table to several classes. This could be the case when a database is not normalized and redundant information is present. In this case, different projections of the table could be mapped to different classes.

15. **Write Support**: Usually, RDF data are accessed in a read-only manner. Nevertheless, a mapping language should provide support for write access to the data. This feature explicitly supports write access to the relational data, based on the produced RDF graphs.

In [19], mapping languages are classified in four categories. Direct Mapping, Read-only general-purpose Mapping, Read-write general-purpose mapping and special-purpose

mapping. The direct mapping is a simple approach for transforming relational tables into RDF, thus most of the desired features presented above are not supported.

Read-only general-purpose mappings (R2RML, D2RQ and Virtuoso) provide similar functionality with some minor variations (e.g., support for named graphs and static metadata). The expressiveness of these mapping languages makes them suitable for a great variety of applications where the transformation of a relational database to a read-only graph is needed.

Read-write general-purpose mappings (R3M) focus on providing bidirectional access to the data stored in a relational database. This feature comes at the expense of poorer support for logical table mappings (F1) and selection conditions (F4) due to the view update problem[5]. As a result, the expressiveness of the read-write general-purpose mapping is less expressive than read-only general-purpose mappings in order to guarantee support for update operations on the underlying relational data.

Special-purpose mapping languages (R_2O, Triplify) were not designed to be highly reusable like the general-purpose mappings presented above, but were driven by use-cases. Thus, not all of them provide support for static metadata, named graphs or blank nodes. However, these mapping languages are appropriate for applications that are similar to the respective use-cases.

R2RML and D2RQ support most of the features described above and can be used to produce automatically a direct mapping from a relational database, while the produced mapping can also be based on a pre-existing vocabulary. This feature makes them ideal for applications that consume linked data, in existing ontologies can be reused or new ones may be developped for modeling Earth Observation data. R2RML and D2RQ also support several transformation functions on the values of the database which may be a highly desired features. However, R2RML and D2RQ do not support update operations on the underlying relational data. Therefore, the expressivenes of R2RML and D2RQ make them the most suitable languages and tools for expressing and processing mappings for publishing data, that can be consumed by applications (e.g., use case of project LEO).

2.3 The mapping language R2RML

R2RML is a language for expressing customized mappings from relational databases to RDF datasets. Such mappings provide the ability to transform existing relational data into the RDF data model, expressed in a structure and target vocabulary of the mapping author's choice. R2RML mappings are themselves RDF graphs usually encoded in the Turtle

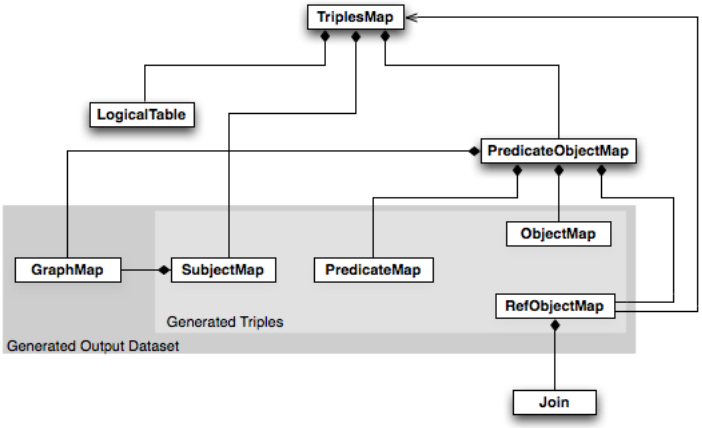

Figure 2.1: An overview of R2RML constructs

syntax. R2RML allows different types of mapping implementations. An application that processes R2RML mappings, also known as an R2RML processsor, can choose to leave the relational database intact and provide a virtual SPARQL[4] endoint that transparently translates SPARQL queries to SQL queries by taking into account the given R2RML mappings. Alternatively, an R2RML processor can choose to export all relational data as an RDF dump according to the given mappings and then offer a linked data interface over the produced data [28].

Every R2RML mapping is tailored to a specific database schema and target vocabulary. The input to an R2RML mapping is a relational database that conforms to that schema. The output is an RDF dataset, as defined in SPARQL, that uses predicates and types from the target vocabulary. The mapping is conceptual; R2RML processors are free to materialize the output data, or to offer virtual access through an interface that queries the underlying database, or to offer any other means of providing access to the output RDF dataset.

2.3.1 Structure of an R2RML mapping

R2RML mappings refer to logical tables to retrieve data from an input database. A *logical table* is a relational table that is explicitly stored in the database, an SQL view or a valid SQL select query. For each logical table that will be exported into RDF, a triple map has to be defined. A *triple map* is a rule that defines how each tuple of the logical table will be mapped to a set of RDF triples. Each triple map consists of two parts.

An R2RML mapping consists of a subject map and one ore more predicate-object maps.

[4] `http://www.w3.org/TR/2008/REC-rdf-sparql-query-20080115/`

A *subject map* forms the subject of all RDF triples generated from a logical table row. Usually, the primary key of the table is used to generate automatically the IRI of the resource that will appear on the subject part of all generated triples. 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 column of the logical table specified.

In order to produce RDF triples, one has to combine the subject map with a predicate map and an object map and apply all of them to each tuple of the related logical table. By default, all triples can be assigned to the default graph of the output RDF dataset, or to a named graph. For this purpose, a graph map can be declared in the subject map or the predicate-object map. A *graph map* defines the URI of the named graph that will contain the produced RDF triples.

According to the R2RML specification, subject maps, predicate maps, object maps and graph maps are term maps. A *term map* is a function that generates an RDF term from a row of a logical table. Term maps are used to generate RDF terms from existing data and are allowed to define functions that returns constantly the same RDF term, functions that return an RDF term according to the contents of a row of a logical table, and RDF terms that are produced automatically by a templating mechanism.

In order to express relationships among logical tables, R2RML defines referencing object maps. A *referencing object map* is defined within a predicate-object map and defines the relationship that holds between the objects to be generated and the subject of another triple. Since this relationship can be referencing different logical tables, a join may be defined between the two logical tables using a join condition. A *join condition* is an RDF resource that has a parent and child properties that define the column names of the respective logical tables. Figure 2.1 was originally presented in [28] and depicts the relationship between the aforementioned constructs that constitute an R2RML mapping.

2.3.2 Useful features of R2RML

R2RML is not limited to mapping relational tables to RDFS classes and relational attributes to data properties. R2RML goes beyond the direct mapping approach and has several useful features that are presented below:

- **Ad-hoc SQL result-sets**: A logical table is not obliged to be a physical table in the database but it can be an SQL view or the result of an SQL select query. This

feature is very 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 $Employee(\underline{id}, name, surname, salary)$. 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 contraints that may exist in the underlying relational database to make these 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 caters for this case and allows a user to customize a subject map so that produced triples can belong to the default graph or any other named graph.

2.4 Processors of R2RML mappings

An R2RML processor is a software application that supports all the features of R2RML, namely generating a mapping from a database and produce a respective RDF dump file. We will now present in some detail several R2RML processors, namely OpenLink Virtuoso, RDF-RDB2RDF, XSPARQL, UltraWrap, DB2Triples, Morph, and D2RQ.

2.4.1 OpenLink Virtuoso

OpenLink Virtuoso[5] is a high-performance object-relational SQL database. Virtuoso defines its own proprietary mapping language, nick-named Linked Data Views, which uses Virtuoso's Meta Schema Mapping Language to map relational data to RDF. Linked Data Views is similar to R2RML, so R2RML support is achieved by the inclusion of a simple translator

[5] `http://virtuoso.openlinksw.com`

which translates R2RML mappings to Linked Data Views. However, due to several limitations of the optimizer of the native Linked Data Views implementation of Virtuoso, the `rr:sqlQuery` feature of R2RML is not supported, meaning that logical tables derived from SQL queries cannot be mapped into RDF graphs.

2.4.2 RDF-RDB2RDF

RDF-RDB2RDF[6] is a processor of R2RML mappings. It is a Perl application that contains modules that support both the Direct Mapping mechanism, and the R2RML mapping language. RDF-RDB2RDF takes as input an R2RML mapping and generates an RDF dump. The R2RML module also implements an extension of R2RML that allows the dynamic definition of a language tag that takes into account the contents of a column.

2.4.3 XSPARQL

XSPARQL[7] is a prototype implementation of the XSPARQL language for mapping between XML and RDF in either direction. XSPARQL is a rewriter that translates an XSPARQL query into an XQuery. It supports the RDB2RDF Direct Mapping and R2RML mapping language and currently can access relational data stored in MySQL and PostgreSQL.

2.4.4 Ultrawrap

Ultrawrap[8] is a tool developed by Capsenta. Ultrawrap is not a native RDF store, but provides a publishing and querying interface. The publishing interface transforms the contents of an underlying relation database into RDF by processing direct mappings, R2RML mappings or D2RQ mappings. The querying interface employs a simple translation algorithm into SQL, without implementing any optimizations. It also provides a graphical user interface for customizing mappings (e.g. exclude a table column or give a specific URI pattern for the instances). Figure 2.2 displays the architecture of this tool.

[6] `https://metacpan.org/release/RDF-RDB2RDF`
[7] `http://xsparql.deri.org`
[8] `http://capsenta.com/ultrawrap/`

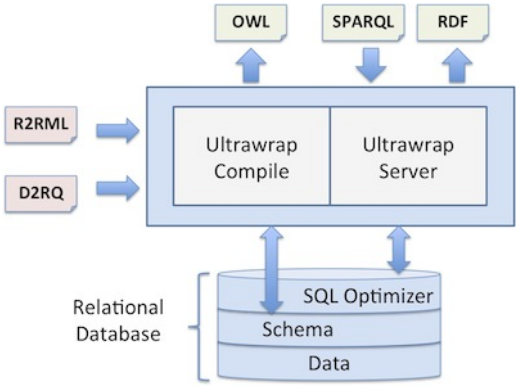

Figure 2.2: Architecture of Ultrawrap

Ultrawrap consists of two parts: Compile and Server. Ultrawrap Compile processes the mappings and generates RDF graphs. Ultrawrap Server is responsible for the execution of SPARQL queries by translating them to SQL, thus pushing all optimizations to the SQL engine.

2.4.5 db2triples

The tool db2triples[9] is an R2RML processor that also provides support for the direct mapping approach. Its functionality is limited to accepting an R2RML mapping file and the details for connecting to a relational database, and then produces RDF triples in a desired format (e.g. TURTLE, N3). However, it is not able to generate automatically R2RML mappings or evaluate SPARQL queries over the underlying relational database.

2.4.6 Morph

Morph[10] is an RDB2RDF engine which implements the R2RML specification. Currently, it supports two operations: the generation of an RDF graph from relational data given a valid R2RML mapping, and SPARQL query processing. SPARQL queries are evaluated over virtual RDF graphs, by translating them into SQL according to the input R2RML mapping. Several optimizations are employed during the translation phase, such as self-join elimination, subquery elimination and left outer-join elimination.

Several algorithms have been implemented for translating SPARQL into SQL based on predefined mappings. Especially for the case of R2RML mappings, usually such algorithms

[9] `https://github.com/antidot/db2triples`
[10] `https://github.com/jpcik/morph`

are not satisfactory enough in terms of performance (e.g. query evaluation time). The latest release of Morph [26], extents Chebotko's algorithm for translating SPARQL to SQL by taking into account R2RML mappings. The performance of this algorithm was evaluated by comparing native SQL queries (hand-written), naive-translation queries generated by the modified Chebotko's algorithm, and three variants of the latter, being subquery elimination (SQE), self-join elimination (SJE) and a combination of both (SQE + SJE). It is observed in the results that overall, the performance of the translated queries was similar. Morph is also evaluated through the analysis of real world queries, from projects like BizkaiSense, REPENER and Integrate, by comparing the response time between D2R and Morph.

2.4.7 D2RQ

The D2RQ[11] platform is a system for accessing relational databases as virtual, read-only RDF graphs. Similarly to Ultrawrap and Morph, it offers RDF-based access to the content of a relational database, without having to replicate its contents into an RDF store. The main operations supported by D2RQ are the following:

- Generate mappings from an input relational database. Such mappings can be written either in the D2RQ mapping language or R2RML.
- Create RDF graphs, serialized in several RDF formats (e.g. TURTLE, N3, RDF/XML), from the content of the relational database by taking into account appropriate mappings.
- Query a non-RDF database using SPARQL. SPARQL queries are translated to SQL using the appropriate mappings.

The developers of D2R originally defined the D2RQ mapping language which describes mappings between a relational database and an ontology. The language D2RQ has been described in some detail in Section 2.1. Apart from the D2RQ language, the D2R platform provides support for R2RML mappings. The platform consists of the following components: the D2RQ engine and the D2R server. The D2RQ engine, which is a plug-in for the Jena Semantic Web toolkit[12], uses the mappings to rewrite Jena API calls to SQL queries that are evaluated against the underlying database. The D2R server is an HTTP server that provides a linked data view for the contents of the underlying database and a SPARQL endpoint. Figure 2.3 presents the architecture of the D2RQ platform. Finally, the D2RQ

[11] `http://d2rq.org/`
[12] `http://semanticweb.org/wiki/Jena_(toolkit)`

platform supports many popular relational databases such as MySQL, PostgreSQL, Oracle, SQL Server, HSQLDB and other database systems compliant with the SQL-92 standard.

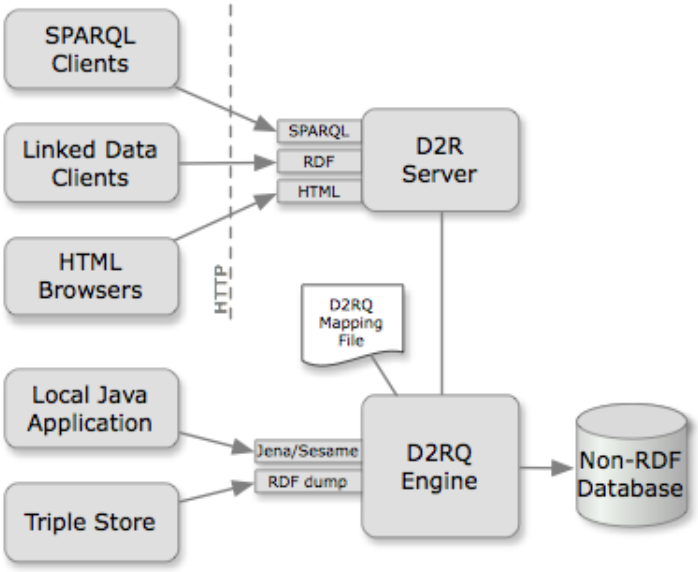

Figure 2.3: Architecture of the D2RQ Platform

2.5 Summary

In this chapter, we presented the state of the art RDB-to-RDF mapping languages designed for mapping relational data to RDF. We presented in more detail the R2RML mapping language which recently became a W3C recommendation and presented implemented systems that are based on R2RML mappings.

Chapter 3
Transforming Earth Observation data into RDF graphs

Earth Observation data can come in various forms. In this chapter we will discuss in some detail the formats that are most commonly used in the domain of Earth Observation for representing and exchanging relevant information. Since EO data is geospatial data, we will also discuss related work on transforming geospatial information to RDF graphs.

3.1 Earth Observation data

Earth Observation data can come in vector or raster form usually accompanied by metadata. Vector data, available in formats such as ESRI shape files, 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), 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., SciQL developed by CWI in project TELEIOS. Metadata about EO data are encoded in various formats ranging from custom XML schemas to domain specific standards like the OGC GML Application schema for Earth Observation products and the OGC Metadata Profile of Observations and Measurements. Let us now present in more detail the file formats that are commonly used in the Earth Observation domain.

ESRI Shapefiles store thematic and geometric information for spatial features in a dataset. The ESRI shape file data format[1] is developed and regulated by ESRI as an open specification for exchange geometric information. Thematic information is stored in a dBASE format file (.dbf) which is a relational table. Each tuple of this table is associated to a single geometric object that is stored in the main file (.shp). Information about the coordinate reference system that has been used for encoding the coordinates of the geometric objects is stored in a separate file (.prj). Optionally, a spatial index that may be used for optimizing access

[1] `http://www.esri.com/library/whitepapers/pdfs/shapefile.pdf`

to the geometric information is stored in a separate file (.sbn). A shape file is restricted to contain a single type of geometry for all features. A geometric object may be a point, a line string, a polygon, a collection of points, a collection of line strings, or a collection of polygons, that optionally may contain a measurement value.

Keyhole Markup Language (KML) is an XML grammar for encoding and exchanging geographic data with a main focus on visualization of such data in an earth browser. It was originally developed by Google Inc. and subsequently it was submitted to the Open Geospatial Consortium for becoming an implementation standard. OGC KML 2.2[2] is now an OGC implementation standard. KML focuses not only to the encoding and transportation of geographic information, but also on the representation of such data. Geometric information in KML uses the GML Coordinate Reference System with identifier LonLat84_5773. KML defines appropriate structures for representing points, open and closed line strings, polygons, and collection of geometries that allows the association of more than one geometries to the same feature. The KML specification also defines the appropriate vocabulary for stating whether a 3D browser should drape a geometry over the terrain, whether to connect a geometry to the ground, and how to interpret the altitude components of a geometry. KML also allows the specification of the location, orientation and scale of a textured 3D object. A KML document has various non-spatial attributes like name, visibility, author, link, style, and region, and optionally contain groups of features, simple or complex types that derive from `xsd:anySimpleType`. Features can be described using non-spatial attributes like name, visibility, author, phone number, link, description, and style. The KML standard also caters for the case that user-defined data need to be present for representing a feature.

The Geography Markup Language (GML) is an XML-based encoding standard[3] developed by the OGC for the representation of geospatial data. GML provides XML schemas for defining a variety of concepts that are of use in Geography: geographic features, geometry, coordinate reference systems, topology, time and units of measurement. Initially, the GML abstract model was based on RDF and RDFS, but later the consortium decided to use XML and XML Schema. The GML *profiles* are logical restrictions of GML that might be of use by applications that do not want to use the whole of GML. GML profiles can be specified through an XML document, an XML schema, or both. Some of the profiles that have been proposed for public use are the Point Profile, which defines a simple point geometry in GML and the GML Simple Features Profile, which is the GML encoding of Simple Features for SQL [4]. In addition various GML application schemas have been defined. Applications schemas are XML vocabularies that are application- specific and can be built on specific

[2] `http://portal.opengeospatial.org/files/?artifact_id=27810`
[3] `http://portal.opengeospatial.org/files/?artifact_id=20509`

GML profiles or use the full GML specification.

GeoJSON[4] is a format for encoding a variety of geographic data structures. It contains GeoJSON objects which can represent geometric objects, features or feature collections. Properties of GeoJSON objects are represented using name-value pairs. Features have a property named `geometry` which contain a geometric object. A geometric object can be a point, a collection of points, a linestring, a collection of linestrings, a polygon, a collection of polygons or a collection that contains any of the aforementioned geometric object types.

The Network Common Data Form (netCDF) is a binary data format for array-oriented scientific data. Currently, the netCDF format is a candidate encoding standard[5]. netCDF was designed for representing digital geospatial information that represents space and time-varying phenomena. netCDF data are self-describing as all information about the contents of a file (e.g., spatial and temporal properties of the data) is embedded in the file. netCDF files are used extensively for exchanging information related to forecast models, weather stations, satellites imagery, etc.

Catalogue web services allow the discovery of Earth Observation collections and products and their accompanying metadata. Catalogue Services for the Web (CSW) is an OGC standard [2] for exposing a geospatial catalogue on the Web. CSW defines a common interface for discovering, browsing and querying metadata. A CSW catalogue contains information for geospatial data, geospatial services and other auxiliary resources. Metadata are encoded in XML documents that contain a minimum set of elements like title, format, type (e.g., dataset, collection of datasets or service), bounding box, and coordinate reference system. The standard defines XML schemas for the documents that are exchanged between the client and the catalogue when the client wants to discover the capabilities of the catalogue, when she wants to formulate search requests to the catalogue, and when the server returns the results to the client.

OpenSearch[6] is an ongoing initiative for the definition of a collection of formats for sharing search results. The standard defines the structure of an XML document that informs prospective clients about the capabilities of the search engine and a set of parameterized URL templates that the client can instantiate to make search requests. OpenSearch also defines the envelope structure (as usually the matching records can be obtained in various formats) of response XML documents that contains the results of a search request.

[4] `http://geojson.org/geojson-spec.html`
[5] `http://www.opengeospatial.org/standards/netcdf`
[6] `http://www.opensearch.org/`

3.2 Transforming geospatial data into RDF graphs

Up to now, little attention has been paid to the problem of publishing geospatial information in the Semantic Web. Most datasets that contain geospatial information are either generated manually or by semi-automated processes. In this section we review related work in publishing geospatial information in RDF.

LinkedGeoData (LGD)[7] is a project focused on publishing OpenStreetMap (OSM)[8] data as linked data. OSM maintains a global editable map that depends on users to provide the information needed for its improvement and evolution. The respective ontology is derived mainly from OSM tags, i.e., attribute-value annotations of nodes, ways, and relations. The tool Sparqlify[9] has been recently developed by the same group, for mapping relational data to RDF and linked data. Sparqlify is based on a custom mapping language (nicknamed Sparqlify mapping language), which resembles SPARQL construct queries and SQL statements for defining views. The Sparlify mapping language allows the mapping of spatial datatypes into RDF but does not discuss how one can deal with data that are not stored in relational databases.

In [12], the authors present how R2RML can be combined with a spatially enabled relational database in order to transform geospatial information 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 preprocessing (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, which is in contrast to other approaches discussed earlier in this chapter. However, in this work, no automated method for publishing geospatial datasets into RDF is discussed.

The tool geometry2rdf[10] allows the transformation of geospatial information stored in relational databases into RDF. Currently, geometry2rdf supports only the Oracle Spatial relational database and utilizes the Jena and Geotools libraries for generating the RDF data. The tool follows the direct mapping approach, which is not expressive enough to deal with the specificities of the Earth Observation domain.

The tool TripleGeo [25] developed in project GeoKnow[11] is an open source Extract-Transform-Load (ETL) utility designed to publish into RDF data stored in a spatially en-

[7] `http://linkedgeodata.org/`
[8] `http://www.openstreetmap.org/`
[9] `http://sparqlify.org/`
[10] `http://mayor2.dia.fi.upm.es/oeg-upm/index.php/en/technologies/151-geometry2rdf`
[11] `http://geoknow.eu/`

abled database or in an ESRI shapefile. TripleGeo is an extension of geometry2rdf which adds support for the GeoSPARQL vocabulary. However, it also follows the direct mapping approach which is not as expressive as R2RML which is a general purpose mapping language.

3.3 Summary

In this chapter we presented the formats that are most commonly used in the domain of Earth Observation for representing and exchanging geospatial information. We also presented related work on transforming geospatial information to RDF graphs.

Chapter 4
GeoTriples: a tool for transforming Earth Observation data into RDF graphs

The previous two chapters discussed previous research on transforming relational and geospatial data sources into RDF. We discussed W3C standards which involve mapping languages to create user defined mappings between an input database source and the elements of a target RDFS schema. One critical problem of these mappings is that they do not handle most popular geospatial data types, and do not utilize state-of-the art data models and query languages for linked geospatial data (stSPARQL and GeoSPARQL). In this thesis we are developing the tool GeoTriples that will be able to transform geospatial data sources into RDF data modelled as in the recent works on stSPARQL and GeoSPARQL. We are going beyond the related work on mapping relational databases to RDF graphs by designing and developing a tool that will be able to translate a rich variety of geospatial data sources into RDF according to the relevant OGC standards. We show how GeoTriples can produce automatically R2RML mappings for Earth Observation data stored in spatially-enabled databases or ESRI shape files and process such mappings in order to produce RDF graphs that utilize the state-of-the art data models (stSPARQL and GeoSPARQL). Other kinds of EO data will be dealt as they will be needed in the course of the project.

The organization of this chapter is as follows. In Section 4.1 we present the conceptual architecture of GeoTriples, in Section 4.2 we present the systems D2RQ that we chose to use as the basis of our implementation. Finally, in Section 4.3 we present some implementation details for GeoTriples, where we discuss how GeoTriples performs the automatic generation of R2RML mappings and how it processes them in order to produce RDF graphs.

4.1 The Architecture of GeoTriples

Earth Observation data sources involve vector, structured, and raster data. Vector and structured data may be found in file formats, such as ESRI shape files, KML documents, or GeoJSON documents. Likewise, raster data may be found in image file formats, such as GeoTIFF or netCDF. Using GeoTriples, we will publish the former kind of data, namely vector

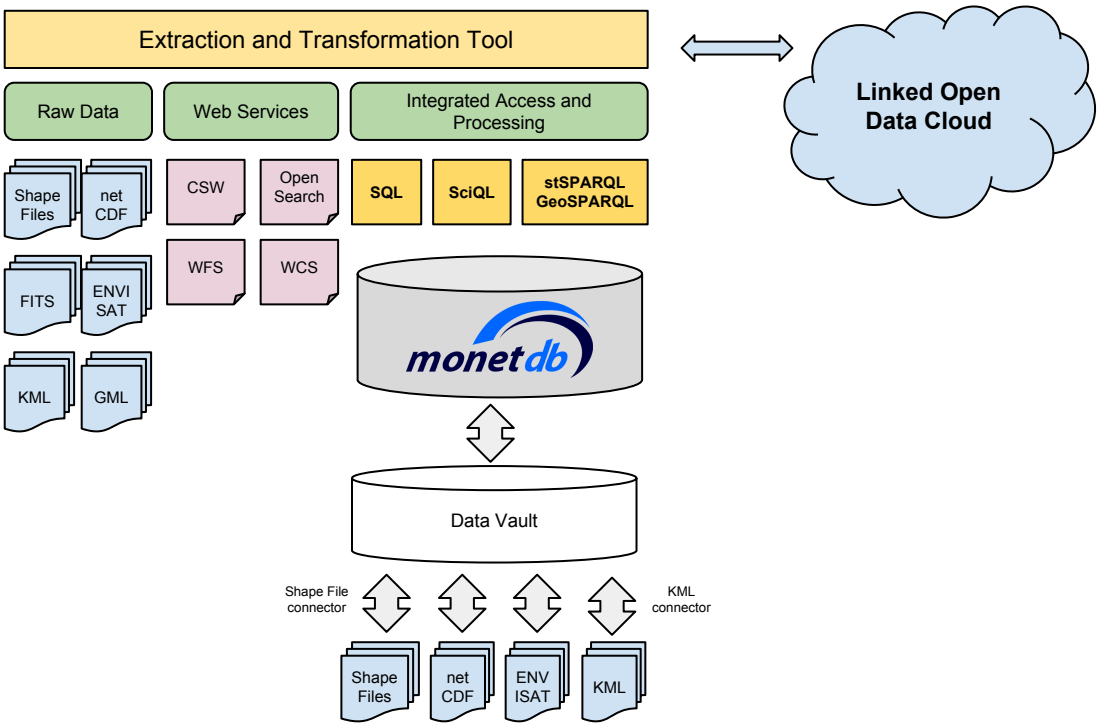

Figure 4.1: The abstract architecture of the system GeoTriples

and structured, as RDF graphs using the vocabulary of the GeoSPARQL query language. GeoSPARQL is a recent OGC standard for representing and querying geospatial information in the Semantic Web, thus supporting its vocabulary should be an essential requirement of any publishing tool. Apart from the file formats mentioned above, we are also interested in publishing as linked data the contents of catalogues that are widely used in the Earth Observation domain as Catalogue Service for the Web (CSW) and Open Search registries. Finally, the tool will also be able to publish as linked data the results of an SQL, SciQL and stSPARQL/GeoSPARQL query over such data.

Figure 4.1 displays the abstract architecture of the tool GeoTriples from a data perspective. The tool will support several types of data formats as input. These types include raw data, output data of web services, data stored in relational databases as well as the results of processing over data stored in a relational database. Raw data include formats like ESRI shape files and KML documents as well as raster formats like GeoTIFF and netCDF image files.

The system architecture of GeoTriples is presented in Figure 4.2. The connector component is responsible for providing an abstraction layer that allows the components of GeoTriples to transparently access input data. GeoTriples comprises two main components: The mapping generator and the R2RML processor. Given an input data source, the map-

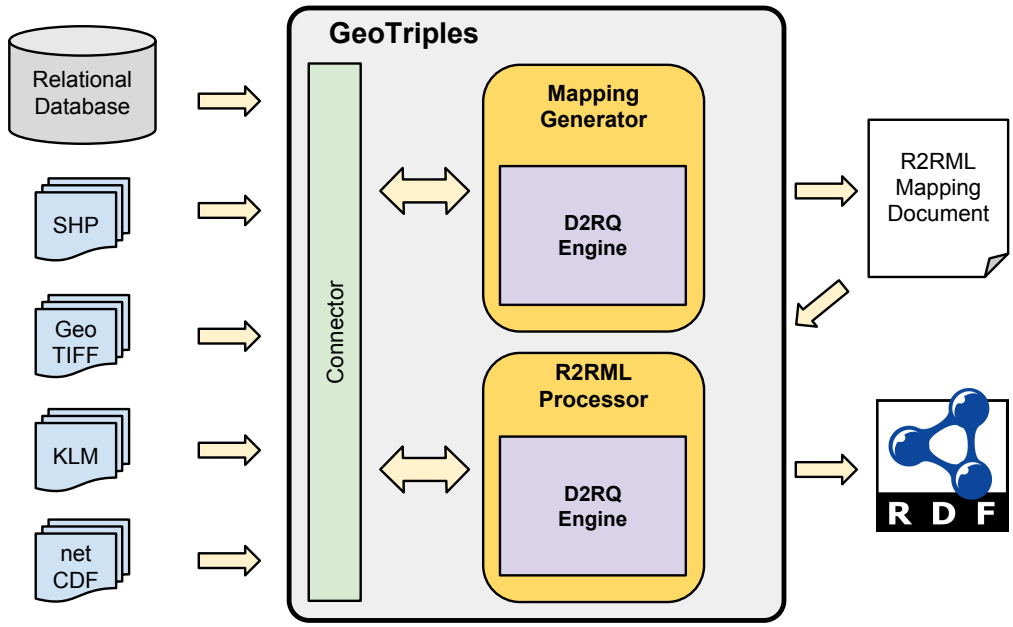

Figure 4.2: The system architecture of GeoTriples

ping generator creates automatically an R2RML mapping that transforms the input data into data in the RDF data model. The produced R2RML mapping is enriched with subject and object maps that take into account the specificities of geospatial data by using the GeoSPARQL vocabulary as the target vocabulary. The R2RML processor of GeoTriples is responsible for generating RDF graphs from the input data by taking into account the mapping documents produced by the mapping generator.

4.2 Implementation Choices

In this section, we discuss the implementation choices we made while designing GeoTriples. We discuss which system we choose for the mapping generator and the R2RML processor components of GeoTriples, as well as why we choose MonetDB as the main relational backend for accessing scientific files repositories and evaluating queries over multi-dimensional arrays.

4.2.1 The D2RQ platform and Morph

One of the implementation choices we had to make during the design of GeoTriples was to choose which RDB2RDF framework to extend. We examined all the discussed in Section 2.4 alternatives, among which were OpenLink Virtuoso, DB2triples and D2RQ. D2RQ was one of

the most mature systems for publishing relational data into RDF that we tested. It provides a mechanism for generating R2RML mappings by accessing the catalogue of a relational database which is a highly desired feature. D2RQ also supports several relational databases such as Oracle, MySQL, PostgreSQL, and MonetDB. D2RQL also provides support evaluating SPARQL queries on a relational database by translating the SPARQL queries into SQL queries taking into account the existing mappings. However, given the recent results on the performance of the system Morph [26] in the future, we will also consider in the future the possibility of utilizing Morph as the R2RML processor of GeoTriples in addition to the D2RQ engine.

4.3 Transforming geospatial data into RDF graphs

In this section we will present the main functionality of GeoTriples. First, we will present how GeoTriples generates automatically R2RML mappings given a spatially-enabled database such as PostGIS or an ESRI shape file. Then, we will discuss the process of generating an RDF graph from the input data by processing these R2RML mappings.

4.3.1 Automatic generation of R2RML mappings

Much work has been done recently on extending RDF for representing and querying geospatial information. The most mature results of this work is the data model stRDF and the query language stSPARQL [22, 23] and the OGC standard GeoSPARQL [24]. We will briefly present the vocabulary defined in GeoSPARQL since we will use it for the representation of geospatial information.

GeoSPARQL is an OGC standard for the representation and querying of geospatial linked data. GeoSPARQL defines much of what is required for such a query language by providing vocabulary (classes, properties, and functions) that can be used in RDF graphs and SPARQL queries to represent and query geospatial data. Figure 4.3 presents a part of the GeoSPARQL ontology. The top level classes defined in GeoSPARQL are `geo:SpatialObject` and `geo:Feature`. `geo:SpatialObject` is the top class and has as instances everything that can have a spatial representation. `geo:Feature` represents all the features and is the superclass of all classes of features users might want to define. `geo:SpatialObject` is a superclass of `geo:Feature` and `geo:Geometry`. Additional vocabulary is defined for asserting and querying information about geometries.

The focus of R2RML is to map the contents of a relational database to RDF graphs,

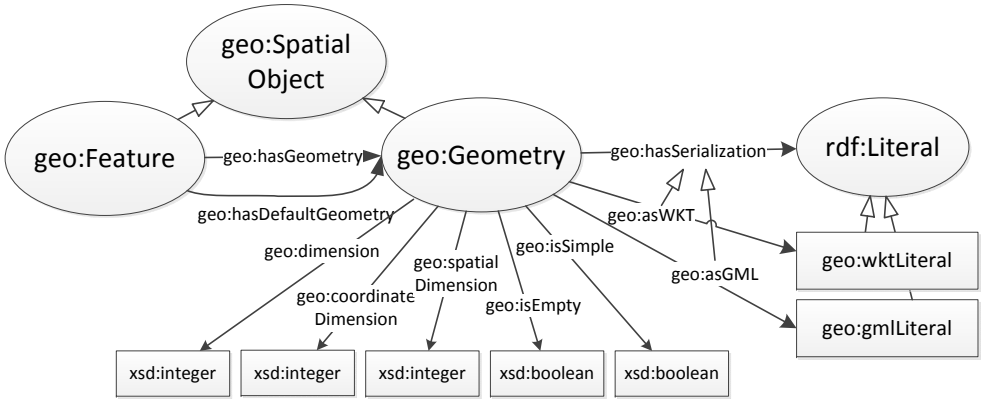

Figure 4.3: Part of the GeoSPARQL ontology

while in the Earth Observation domain we have to allow additional types of data sources as input. For this reason, we will extend R2RML with logical geometry tables. We define a *logical geometry table* as a view over a logical table that contains a geometry column. The logical geometry table contains the columns comprising the primary key of the logical table and the serialization of the geometry along with metadata defined by the GeoSPARQL vocabulary. In terms of the GeoSPARQL vocabulary, the logical table will be used for populating instances of the `geo:Feature` class, while the logical geometry table will be used for populating instances of the `geo:Geometry` class. An appropriate `rr:predicateObjectMap` has to be defined in order to link instances of the `geo:Feature` and `geo:Geometry` class using the appropriate join condition on the primary key of the logical tables and the logical geometry tables. Figure 4.4 depicts a logical table, a logical geometry table and the links between them. The logical table consists of the primary and thematic attributes describing a feature, while the logical geometry table consists of the primary key of the logical table, and a set of geometric attributes that derive by applying geospatial functions like `geof:asWKT` and `geof:isSimple` to the stored geometry. This allows us to create mappings that map information that exists implicitly in the input data but is required to be present in the RDF graph in order to be compliant with the GeoSPARQL vocabulary.

Let us now demonstrate how one could employ our extension of the R2RML mapping language with logical geometry tables to transform the Natura 2000 dataset, which is distributed as an ESRI shape file, into RDF.

Example. Natura 2000 is an ecological network designated under the Birds Directive and the Habitats Directive of the European Union which together form the cornerstone of Eu-

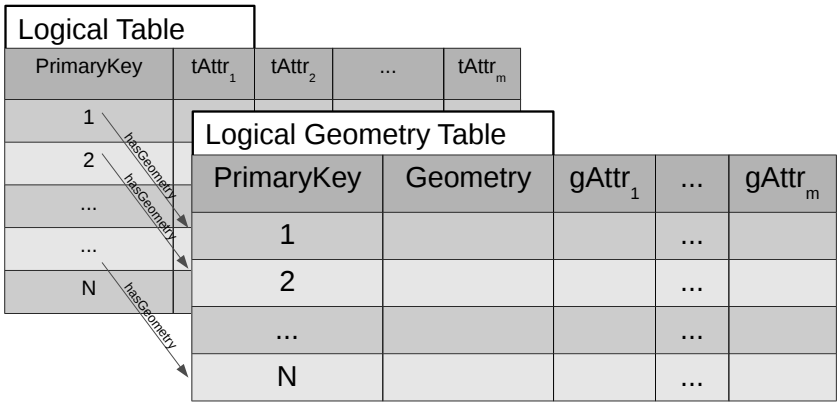

Figure 4.4: An example of a logical table and the corresponding logical geometry table

rope's nature conservation policy. We present the Natura dataset in more detail in Chapter 5. The Natura 2000 areas for Germany are shown geographically in Figure 4.5.

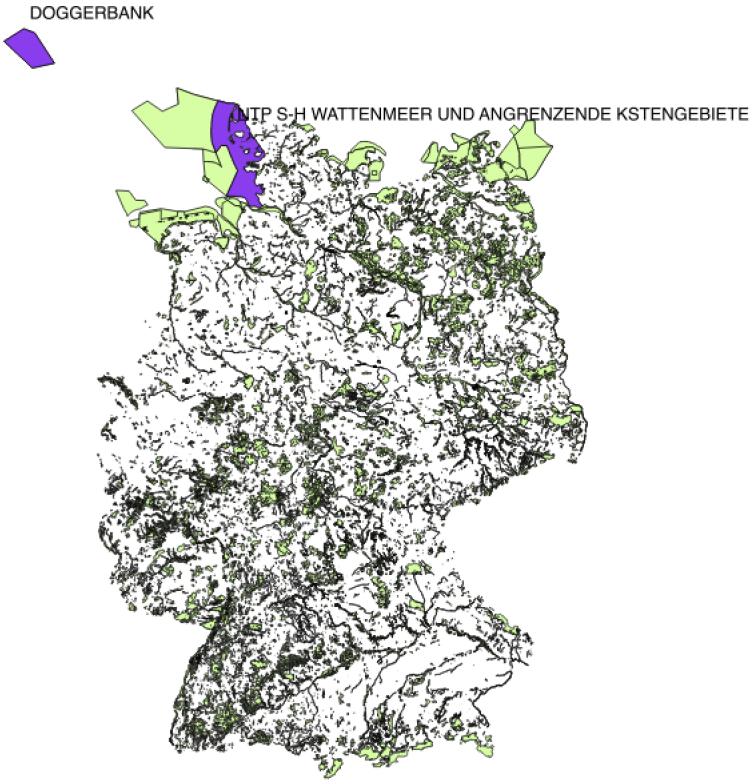

Figure 4.5: Natura 2000 areas for Germany

The Natura 2000 dataset for Germany is distributed in the form of ESRI shape files that contain thematic and geometric information about the protected areas. Each geometric object is characterized by the following thematic attributes:

The contents of the shape file can be modeled as a logical table that coresponds to the relational table with schema `Natura(gid BIGINT, Geom GEOMETRY, SiteCode VARCHAR(255),`

Table 4.1: Attributes schema

Attribute Name	Data Type
SiteCode	String
SiteName	String
Release_Date	Date
SiteType	String
MS	String
Point_X	Double
Point_Y	Double
Shape_Length	Double
Shape_Area	Double

Table 4.2: A sample of the contents of the Natura 2000 dataset

SITECODE	SITENAME	RE-LEASE_DATE	SITETYPE	MS	POINT_X	POINT_Y	Shape_Length	Shape Area
DE0916391	NTP S-H W...	2011-01-27	K	DE	4221691.4	3492908.0	836551.3	4525462405.6
DE1003301	DOGGER-BANK	2011-01-27	B	DE	3953766.1	3623776.9	171212.4	1695526629.1

`SiteName VARCHAR(255), Release_Date DATE, SiteType VARCHAR(255), MS VARCHAR(255), Point_X DOUBLE PRECISION, Point_Y DOUBLE PRECISION, Shape_Length DOUBLE PRECISION, Shape_Area DOUBLE PRECISION)`, where `GEOMETRY` is the geometric data type defined by the "Simple feature access - Part 2: SQL option" OGC standard [3]. In Table 4.2 we present two features from the Natura 2000 dataset for Germany .

The desired RDF triples to be produced from this shape file are as follows:

```
<http://data.example.com/natura/Feature/id/1>
                rdf:type geo:Feature ;
                nato:has_SITECODE "DE0916391"^^xsd:string ;
                nato:has_SITENAME "NTP S-H W..."^^xsd:string ;
                nato:has_RELEASE "2011-01-27"^^xsd:date ;
                nato:has_SITETYPE "K"^^xsd:string ;
                nato:has_MS "DE"^^xsd:string ;
                nato:has_POINT_X "4221691.4"^^xsd:double ;
                nato:has_POINT_Y "3492908.0"^^xsd:double ;
                nato:has_Shape_Length "836551.3"^^xsd:double ;
                nato:has_Shape_Area "4525462405.6"^^xsd:double ;
                geo:hasGeometry
<http://data.example.com/natura/Geometry/id/1> .
```

```
<http://data.example.com/natura/Geometry/id/1>
                  rdf:type geo:Geometry ;
                  geo:dimension "2"^^xsd:integer ;
                  geo:coordinateDimension "2"^^xsd:integer ;
                  geo:spatialDimension "2"^^xsd:integer;
                  geo:isEmpty "false"^^xsd:boolean ;
                  geo:isSimple "true"^^xsd:boolean ;
                  geo:asWKT "POLYGON((...))"^^geo:wktLiteral .
```

Every feature in the shape file becomes an instance of the class `geo:Feature` and each thematic attribute is assigned to it by an RDF triple. Notice that a new instance of the class `geo:Geometry` is created for representing the spatial information of the feature.

For creating such RDF triples we need to introduce the concept of a logical geometry table. As we mentioned earlier, a logical geometry table is a view over the aforementioned relational table, that contains the columns comprising the primary key of the logical table and additional columns that contain the serialization of the geometry along with metadata about the geometry using the GeoSPARQL vocabulary. These metadata represent useful information about geometries such as the dimension of the coordinates of a geometry, whether a geometry is empty or simple etc. The logical geometry table in this example is a table with schema `NaturaGeometryTable(gid BIGINT, dimension SMALLINT, coordinateDimension SMALLINT, spatialDimension SMALLINT, isEmpty BOOLEAN, isSimple BOOLEAN, asWKT VARCHAR(4096))` where the contents of the column `dimension` is the result of applying the function `geof:dimension` to the column `Geom` and so on. The following R2RML mapping produces RDF triples that contain the thematic information of each feature in the shape file.

```
_:natura
    rr:logicalTable [ rr:tableName "‘natura‘"; ];
    rr:subjectMap [
       rr:class geo:Feature;
       rr:template "http://data.example.com/natura/Feature/id/{‘gid‘}"; ];

    rr:predicateObjectMap [
        rr:predicate nato:has_SITECODE;
        rr:objectMap [ rr:datatype xsd:string;
                       rr:column "‘SITECODE‘"; ];
    ];
```

```
    rr:predicateObjectMap [
        rr:predicate nato:has_SITENAME;
        rr:objectMap [ rr:datatype xsd:string;
                       rr:column "`SITENAME`"; ];
    ];
    rr:predicateObjectMap [
        rr:predicate nato:has_RELEASE;
        rr:objectMap [ rr:datatype xsd:date;
                       rr:column "`RELEASE`"; ];
    ];
    rr:predicateObjectMap [
        rr:predicate nato:has_SITETYPE;
        rr:objectMap [ rr:datatype xsd:string;
                       rr:column "`SITETYPE`"; ];
    ];
    rr:predicateObjectMap [
        rr:predicate nato:has_MS;
        rr:objectMap [ rr:datatype xsd:string;
                       rr:column "`MS`"; ];
    ];
    rr:predicateObjectMap [
        rr:predicate nato:has_POINT_X;
        rr:objectMap [ rr:datatype xsd:double;
                       rr:column "`POINTX`"; ];
    ];
    rr:predicateObjectMap [
        rr:predicate nato:has_POINT_Y;
        rr:objectMap [ rr:datatype xsd:double;
                       rr:column "`POINTY`"; ];
    ];
    rr:predicateObjectMap [
        rr:predicate nato:has_Shape_Length;
        rr:objectMap [ rr:datatype xsd:double;
                       rr:column "`ShapeLength`"; ];
    ];
    rr:predicateObjectMap [
        rr:predicate nato:has_Shape_Area;
```

```
        rr:objectMap [ rr:datatype xsd:double;
                       rr:column "‘ShapeArea‘"; ];
    ].
```

The following R2RML mapping produces RDF triples that contain the geometric information of each feature in the shape file.

```
_:naturaGeometry
    rrx:logicalGeometryTable [ rr:tableName "‘natura‘"; ];
    rr:subjectMap [
        rr:class geo:Geometry;
        rr:template "http://data.example.com/natura/Geometry/id/{‘gid‘}"; ];

    rr:predicateObjectMap [
        rr:predicate geo:hasGeometry ;
        rr:objectMap [
            rr:parentTriplesMap _:natura;
            rr:joinCondition [
                rr:child  "gid";
                rr:parent "gid"; ];
        ];
    ];

    rr:predicateObjectMap [
        rr:predicate geo:dimension;
        rr:objectMap [
            rrx:transformation [
                rrx:function geof:dimension;
                rrx:argumentMap (
                    [rr:column "‘Geom‘"]
                );
            ]
        ];
    ];

    rr:predicateObjectMap [
        rr:predicate geo:coordinateDimension;
```

```
        rr:objectMap [
            rrx:transformation [
                rrx:function geof:coordinateDimension;
                rrx:argumentMap (
                    [rr:column "‘Geom‘"]
                );
            ]
        ];
    ];

    rr:predicateObjectMap [
        rr:predicate geo:spatialDimension;
        rr:objectMap [
            rrx:transformation [
                rrx:function geof:spatialDimension;
                rrx:argumentMap (
                    [rr:column "‘Geom‘"]
                );
            ]
        ];
    ];

    rr:predicateObjectMap [
        rr:predicate geo:isEmpty;
        rr:objectMap [
            rrx:transformation [
                rrx:function geof:isEmpty;
                rrx:argumentMap (
                    [rr:column "‘Geom‘"]
                );
            ]
        ];
    ];

    rr:predicateObjectMap [
        rr:predicate geo:isSimple;
        rr:objectMap [
```

```
            rrx:transformation [
                rrx:function geof:isSimple;
                rrx:argumentMap (
                    [rr:column "‘Geom‘"]
                );
            ]
        ];
    ];

    rr:predicateObjectMap [
        rr:predicate geo:asWKT;
        rr:objectMap [
            rrx:transformation [
                rrx:function geof:asWKT;
                rrx:argumentMap (
                   [rr:column "‘Geom‘"]
                );
            ]
    ];
    ].
```

According to the R2RML specification, graph maps, subject maps, predicate maps, and object maps are term maps. A term map is a function that generates an RDF term from a row of a logical table. It is allowed to define functions that return constantly the same RDF term, functions that return an RDF term according to the contents of a row of a logical table, and RDF terms that are produced automatically by a templating mechanism. Notice in the example above that we have extended the definition of an object map by allowing it to be the RDF term obtained by performing a transformation on the source data. Each transformation defines the SPARQL built-in function or the SPARQL extension function to be invoked, using as argument the sequence of RDF terms that are produced by the respective term maps.

4.3.2 Processing R2RML mappings for producing RDF graphs

The R2RML processor of GeoTriples parses the mappings that were created from the previous step, reads the input data and generates RDF graphs. The R2RML mappings presented in the previous section comprises a triples map for generating the RDF data that represents

thematic information for the input data, and a triple that represent the spatial information of the input data.

The triples map that generates thematic RDF information for the input data, defines a logical table that contains the attributes that comprise the primary key along with attributes with thematic information. In the case of a shape file, the primary key is considered to be the row id of each tuple in the dBASE table. This triples map specifies a subject map that defines the structure of the RDF terms that will consist of the subject of all generated triples. By default, a template is used for generating URIs by taking into account the value of the primary key. The triples map also contains multiple predicate-object maps that define the RDF properties to be used for relating the subject and the object of the generated triple, the value of which originates from the value of the thematic attribute of the logical table.

The triples map that generates spatial RDF information for the input data, defines a logical geometry table that contains the attributes that comprise the primary key along with geometric attributes. Similarly to the previous triples map, in the case of a shape file, the primary key is considered to be the row id of each tuple in the dBASE table. This triples map specifies a subject map that defines the structure of the URI that will be generated for identifying resources representing the geometry of the features that were generated in the previous step. The triples map also contains a predicate-object map for each property of the `geo:Geometry` class, as defined by the GeoSPARQL vocabulary. Each predicate-object map defines that such a property is used for relating the subject and the object of the generated triple, the value of which is computed by applying the respective GeoSPARQL function to the input geometry. For example, the following R2RML mapping fragment defines a predicate-object map that will generate triples with predicate `geo:isSimple` and object the result of applying the GeoSPARQL function `geof:isSimple` to the input geometry.

```
rr:predicateObjectMap [
        rr:predicate geo:isSimple;
        rr:objectMap [
            rrx:transformation [
                rrx:function geof:isSimple;
                rrx:argumentMap (
                    [rr:column "‘Geom‘"]
                );
            ]
        ];
    ];
```

The R2RML processor of GeoTriples, processes these mappings and chooses a different way of evaluating the logical transformation functions mentioned above, depending on the type of the input data. For example, if the input data source is a spatially enabled relational database, GeoTriples will translate this operation to an SQL view that applies to the input geometry the `ST_IsSimple` function defined by the OGC-SFA standard [3]. If the input data source is a shape file, then GeoTriples will perform this transformation on the fly using an external library, for example the JTS Topology Suite.

Table 4.3: Prefixes used in the extended R2RML vocabulary

Prefix	URI
rr	http://www.w3.org/ns/r2rml#
rrx	http://www.w3.org/ns/r2rml-ext#
strdf	http://strdf.di.uoa.gr/ontology#
geo	http://www.opengis.net/ont/geosparql#

4.4 Extensions to R2RML

The main goal of GeoTriples is the generation and processing of R2RML mappings capable of handling EO and geospatial data. The geometric properties of GeoSPARQL, the vocabulary we use by default, are evaluated by values that are not explicity given from the source data, such as the dimension of a given geometry. Such values can be derived by applying a transformation function over the geometry, which is usually available from the source data in binary format. As a result, we extended the R2RML mapping language with new classes and properties in order to support transformation functions over some values that maybe be constant or derive from a column of a relational table or a shapefile layer.

4.4.1 Transformation Extension

The transformation operation is declared in an object map, since the latter describes the objects of the generated triples. Specifically, the operation of transforming a value is modeled using the class rrx:Transformation that contains the URI of the function we want to evaluate and at least one argument. The property that assigns a transformation to an object map is `rrx:transfomation`. Consequently, the domain of `rrx:transformation` is the `rr:TermMap`, since `rr:ObjectMap` is a subclass of `rr:TermMap`, and the range is the `rrx:Transformation`.

As a result, an `rr:TermMap` can also be a transformation-valued-map if we use the property `rrx:transformation`, except for being either a constant-valued-term-map[1] or a column-valued-term-map[2] as described in the R2RML official webpage[3].

An `rrx:Transformation` object, in turn, is described by two objects; An `rrx:Function` object that is evaluated with the URI of the transformation function we want to evaluate (e.g. a GeoSPARQL function), via the property `rrx:function`. The second object is an

[1] `http://www.w3.org/TR/r2rml/\#dfn-constant-valued-term-map`
[2] `http://www.w3.org/TR/r2rml/\#dfn-column-valued-term-map`
[3] `http://www.w3.org/TR/r2rml/`

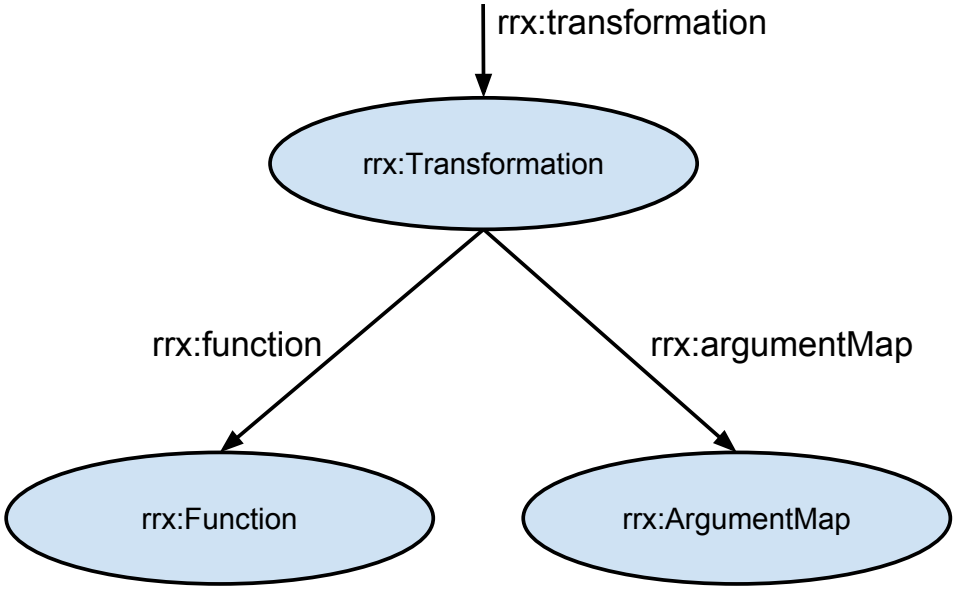

Figure 4.6: Modeling of extension classes and properties

rrx:ArgumentMap object which represents the set of arguments of the subject `rrx:Function`.

The extensions of R2RML that we developed are displayed in Figure 4.6.

Below, we present more formal definitions of the terms we added in the R2RML vocabulary.

4.4.2 Transformation (OWL Class)

Transformation (OWL Class) represents a transformation over a value. A transformation is a function that creates an RDF term from a column of a subject of the logical table described in the subject map. The result of that function is known as the term map's generated RDF term.

Definition

- The URI of this class is `http://www.w3.org/ns/r2rml-ext#Transformation`.
- This class is the domain of the properties `rrx:function` and `rrx:argumentMap`, and the range of property `rrx:transformation`.

Table 4.4: Transformation functions that are supported by GeoTriples

Extension Function URI	Description
`geo:dimension`	Returns the inherent dimension of input geometry
`geo:spatialDimension`	Not supported
`geo:coordinateDimension`	Not supported
`geo:isEmpty`	Returns true if input geometry is an empty geometry. If true, then this geometry represents an empty geometry collection, polygon, point etc
`geo:isSimple`	Returns true if geometry A has no anomalous geometric points, such as selfintersection or selftangency
`geo:asText`	Returns the Well-Known Text (WKT) serialization of input geometry
`geo:asGML`	Returns the Geography Markup Language (GML) serialization of input geometry

4.4.3 Function (OWL Class)

Function (OWL Class) represents the transformation's functionality. An instance of class Function is represented by a resource that has exactly one `rrx:function` property. An `rrx:Function` object is the RDF term that is the value of its `rrx:function` property. In GeoTriples we focalized to spatial transformations over a geometry values and we reuse the existing spatial extension functions that are defined in stSPARQL and GeoSPARQL vocabularies. The currently supported functions are shown in Table 4.4.

Definition

- The URI of this class is `http://www.w3.org/ns/r2rml-ext#Function`.
- This class is the range of the property `rrx:function`.

4.4.4 ArgumentMap (OWL Class)

An `rrx:ArgumentMap` can be either a column-valued-term-map[4] or a constant-valued-term-map[5]. The `rrx:ArgumentMap` is represented by a resource that has exactly one `rr:column`

[4] `http://www.w3.org/TR/r2rml/\#dfn-column-valued-term-map`
[5] `http://www.w3.org/TR/r2rml/\#dfn-constant-valued-term-map`

property, or a `rr:constant` property. A `rrx:Transformation` object refers to a `rrx:ArgumentMap` object via the object property `rrx:argumentMap`.

Definition

- The *URI* of this class is `http://www.w3.org/ns/r2rml-ext#ArgumentMap`.
- The ArgumentMap class is the range of the `rrx:argumentProperty`.

4.4.5 transformation (Property)

The transformation property refers to a rrx:Transformation instance to express the transformed value of a constant or a column's value from a logical table row.

Definition

- The *URI* of this property is `http://www.w3.org/ns/r2rml-ext#transformation`.
- The property is an *object* property.
- The domain of this property is `rr:TermMap`.
- The range of this property is `rrx:Transformation`.

4.4.6 function (Property)

The function property refers to a `rrx:Function` instance.

Definition

- The *URI* of this property is `http://www.w3.org/ns/r2rml-ext#function`.
- The property is an *object* property.
- The domain of this property is `rrx:Transformation`.
- The range of this property is `rrx:Function`.

4.4.7 argumentMap (Property)

The argumentMap property refers to an `rrx:ArgumentMap` instance.

Definition

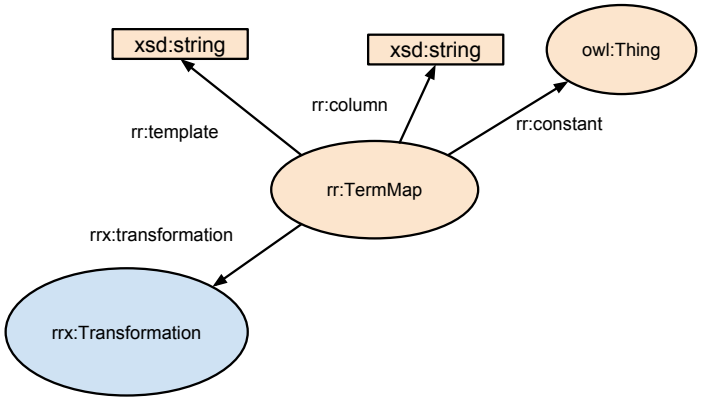

Figure 4.7: Extended rr:TermMap

- The *URI* of this property is `http://www.w3.org/ns/r2rml-ext#argumentMap`
- The property is an *object* property.
- The domain of this property is `rrx:Transformation`.
- The range of this property is `rrx:ArgumentMap`.

4.5 Summary

In this chapter we presented the conceptual architecture of GeoTriples, we presented the systems D2RQ and MonetDB that we chose to use as the basis of our implementation and finally we presented some implementation details for GeoTriples. We discussed how GeoTriples performs the automatic generation of R2RML mappings from spatially-enabled databases or ESRI shape files and how it processes such mappings in order to produce RDF graphs. Other kinds of EO data will be dealt as they will be needed in the course of the project.

Chapter 5
Using GeoTriples in a real world scenario

In this chapter we present how GeoTriples can be used in a real world scenario. We demonstrate how to use GeoTriples to create linked open data that can be exploited in the precision farming domain, as done in project LEO. In this chapter, we briefly describe the ontologies that we developed for representing data that are being produced and then consumed by the precision farming application developed within LEO. Finally, we provide example queries that can be used for data discovery and data manipulation.

5.1 Developing populating ontologies for precision farming

We now present four data sets that will be used in the precision farming application, namely TalkingFields products, Natura 2000 areas, Corine Land Use/Land Cover and the administrative geography of Germany.

5.1.1 Talking Fields products

TalkingFields[1] is a precision farming project funded by ESA and led by LEO partner VISTA (PCA also participates). TalkingFields is an initiative aiming to increase the efficiency of agricultural production via precision farming by means of geo-information services integrating space and ground-based assets. Currently, TalkingFields produces products for improved soil probing using satellite-based zone maps, and provide services for monitoring crop development through provision of biomass maps and yield estimates. Figure 5.1 shows an example of a TalkingFields product with fictional data.

In order to make use of semantic web technologies, we need to transform the data sets provided by TalkingFields to some appropriate format. We now show how a small subset of the provided sets can be represented using this technology. First, we construct an RDFS ontology of concepts and properties depicted in Figure 5.2. This ontology is presented in more detail in the LEO deliverable D5.2.1 [11] but for the convenience of the reader we will

[1] `http://www.talkingfields.de/`

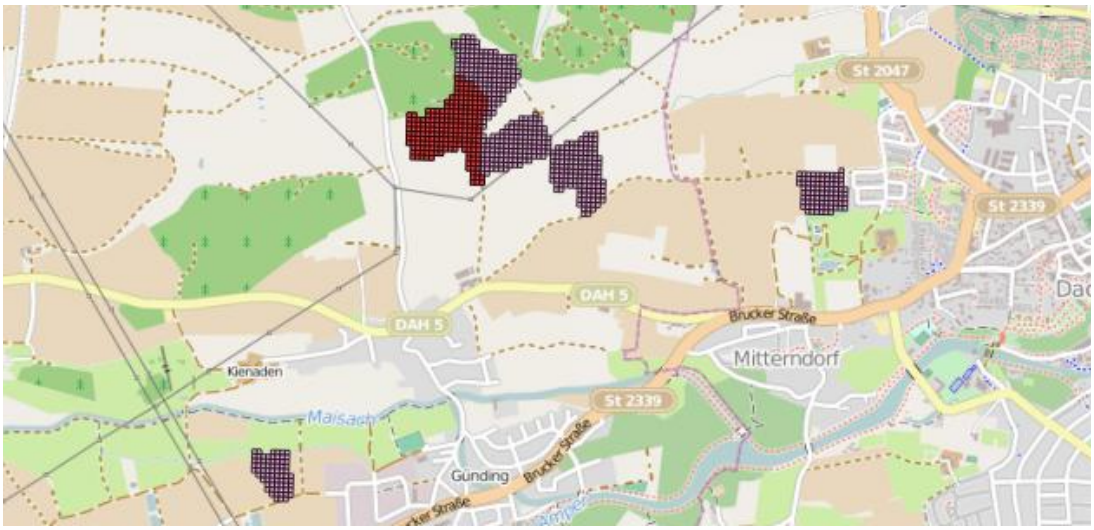

Figure 5.1: Example of a fictional TalkingFields product

now present parts of the developed ontology.

The TalkingFields ontology has three main classes, `tfo:Farm` , `tf:Field` and `tf:RasterCell`. The class `tfo:RasterCell` is the top class of the TalkingFields taxonomy and contains classes that model the hierarchy of agricultural land. The class `tfo:Farm` represents an area of land used for growing crops that contains one or more areas of open land represented by the class `tfo:Field`, which in turn are translated into rectangular land parcels, represented by the class `tfo:RasterCell`.

The main properties of the class `tf:RastelCell` are `tf:hasGeometry` that links a raster cell to its geometric extent, and `tf:hasTFQuality` that defines the quality of a raster cell. The main properties of the class `tf:Field` are `tf:hasHarvest` that represents the quality of the field, and `tf:hasRasterCell` that links a field to the raster cells it consists of. The main properties of the class `tf:Farm` are `tf:hasName` that represents the name of the farm, and `tf:hasField` that links a farm to the fields it consists of.

In Tables 5.1,5.2, and 5.3 we show the contents of an example TalkingFields product. Note that the rows of the table 5.1,5.2, and 5.3 become instances of class `tf:Farm`, `tf:Field`, and `tf:RasterCell` respectively.

Table 5.1: TalkingFields data for farms

FarmID
FarmBetrieb

Next we represent some illustrative part of the TalkingFields datasets using the developed

Table 5.2: TalkingFields data for agricultural fields

FieldID	FieldName	FMArea	Usage	MainPeriod	SubPeriod	Watering	Processed	*FarmID*
1061	16 Wirtsacker	7.1593	1	2012	0	0	1	FarmBetrieb

Table 5.3: TalkingFields data for raster cells

x	y	CV	Vigor	*FieldID*
334200	5411420	12	NULL	1061
334220	5411400	100	2.00	1061
334220	5411340	70.8	-11.00	1061

vocabulary. In what follows we give an RDF representation (in Turtle notation) of a farm that contains a field that consists of three raster cells.

```
tf:Farm1 rdf:type tf:Farm ;
         tf:hasFarmID "FarmBetrieb" ;
         tf:hasField tf:Field1 .

tf:Field1 rdf:type tf:Field ;
          tfo:hasFieldName "16 Wirtsacker" ;
          tfo:hasFMArea 7.1593 ;
          tfo:hasUsage 1 ;
          tfo:hasMainPeriod 2012 ;
          tfo:hasSubPeriod 0 ;
          tfo:hasWatering 0 ;
          tfo:hasPocessed 1 ;
          tfo:hasRasterCell tf:RasterCell1 ;
          tfo:hasRasterCell tf:RasterCell2 ;
          tfo:hasRasterCell tf:RasterCell2 ;

tf:RasterCell1 rdf:type tf:RasterCell ;
               tfo:hasX 334200 ;
               tfo:hasY 5411420 ;
               tfo:hasCV 12 ;
               tfo:hasTFQuality 1 ;
               tfo:hasVigor NULL .
tf:RasterCell2 rdf:type tf:RasterCell ;
               tfo:hasX 334220 ;
```

```
                tfo:hasY 5411400 ;
                tfo:hasCV 100 ;
                tfo:hasTFQuality 0 ;
                tfo:hasVigor 2.00 .
tf:RasterCell3 rdf:type tf:RasterCell ;
                tfo:hasX 334200 ;
                tfo:hasY 5411340 ;
                tfo:hasCV 70.8 ;
                tfo:hasTFQuality -1 ;
                tfo:hasVigor -11.00 .
```

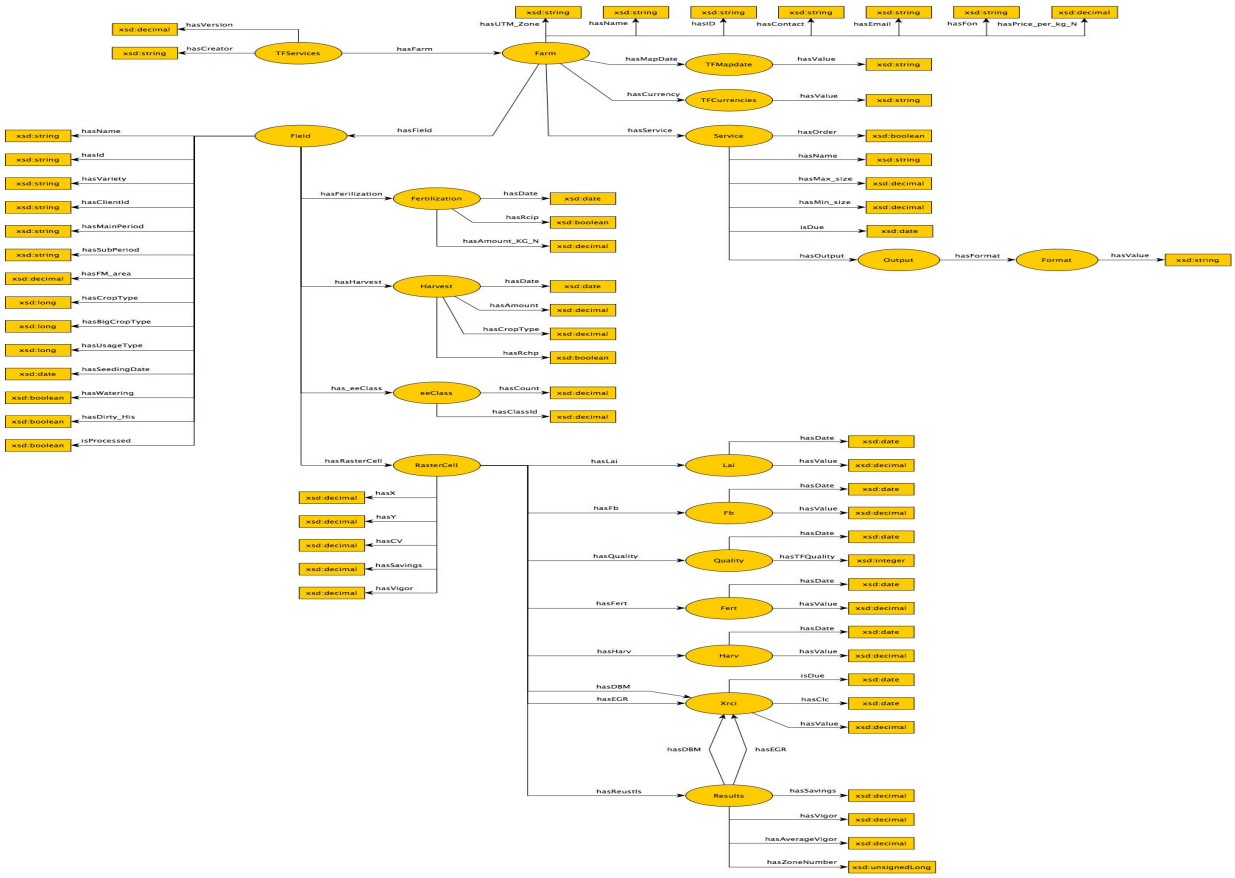

Figure 5.2: Classes in the TalkingFields Ontology

5.1.2 Natura 2000 ontology

Natura 2000 is an ecological network designated under the Birds Directive and the Habitats Directive[2] which together form the cornerstone of Europe's nature conservation policy. National authorities submit a standard data form that describes each site and its ecology in order to be characterized as a Natura site. The spatial extent of each site is submitted by each national authority, and after a validation phase, a spatial database containing all sites is produced. An example of a Natura 2000 area in Germany is presented in Figure 5.3.

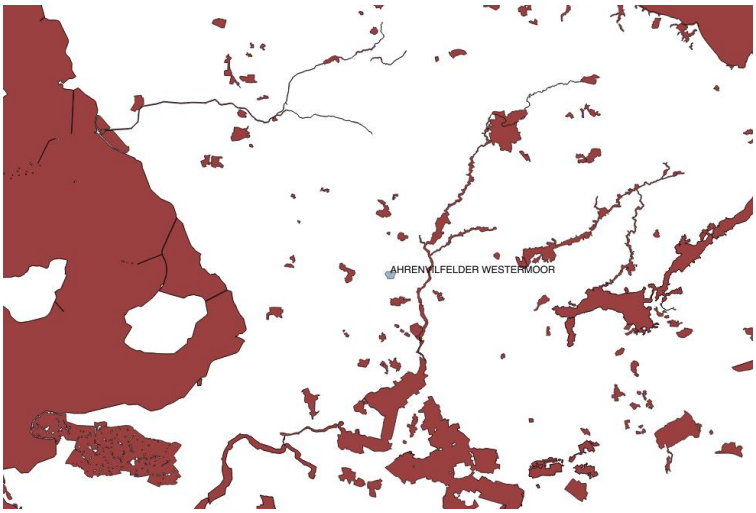

Figure 5.3: Example of a Natura 2000 area

We developed an ontology for the the Natura 2000 dataset that derives from the dataset published by the European Environmental Agency[3] that contains the Natura 2000 sites in Germany. The ontology reuses the three top classes of GeoSPARQL: `geo:Feature`, `geo:Geometry`, `geo:SpatialObject`. Each protected site will be an instance of the class `nat:NaturaArea` that describes features of protected sites. The main properties of the class `nat:NaturaArea` are `nat:hasSiteName` that represents the name of the protected area, `nat:hasSiteType` that represents the type the protected area (e.g., birds directive site), and `geo:hasGeometry` that links a protected area with its geometric serialization.

In Table 5.4 we show an example of a Natura 2000 area.

[2] `http://ec.europa.eu/environment/nature/legislation/habitatsdirective/index_en.htm`
[3] `http://ec.europa.eu/environment/nature/natura2000/access_data/index_en.htm`

Table 5.4: Natura 2000 data

SITECODE	SITENAME	RELEASE_DATE	SITETYPE	MS	POINT_X	POINT_Y	Shape_Length	Shape Area
DE1421304	AHRENVILFELDER WESTERMOOR	2011-01-27	B	DE	4273442.525	3493323.252	3249.00	694680.67

The RDF triples that represent a protected nature reserve in the Ahrenvilfeld community are:

```
nat:field_DE1421304 rdf:type nat:NaturaArea ;
        geo:hasGeometry nat:Geometry_DE1421304 ;
        nato:hasSITENAME "AHRENVI?LFELDER WESTERMOOR"^^xsd:string ;
        nato:hasRELEASE_DA "2011/01/27"^^xsd:string ;
        nato:hasMS "DE"^^xsd:string ;
        nato:hasPOINT_X "4273442.5257"^^xsd:double ;
        nato:hasPOINT_Y "3493323.2525"^^xsd:double ;
        nato:hasShape_Leng "3249.00798832"^^xsd:double ;
        nato:hasShape_Area "694680.677619"^^xsd:double .

nat:Geometry_DE1421304 rdf:type geo:Polygon ;
        geo:hasDefaultGeometry nat:Geometry_DE1421304> ;
        geo:asWKT "<http://spatialreference.org/ref/epsg/32632>
                   POLYGON ((4273436.2628 3493735.5445,
                             4273535.6729 3493720.3748,
                             ... ,
                             4273436.2628 3493735.5445))"^^geo:wktLiteral .
```

5.1.3 Corine Land Cover

The Corine Land Cover (CLC) [4] is an activity of the European Environment Agency that collects data regarding the land use and land cover of European countries. The project uses a hierarchical scheme with three levels of abstraction for describing land cover information. The first level indicates the major categories of land cover on the planet, e.g., agricultural areas, the second level identifies more specific types of land cover, e.g., permanent crops, and

[4] http://www.eea.europa.eu/publications/COR0-landcover

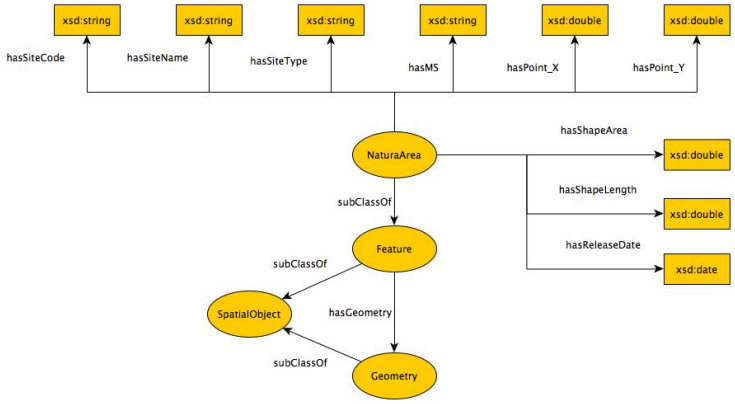

Figure 5.4: Classes in Natura 2000 Ontology

the third level narrows to a specific characterization e.g., fruit trees and berry plantations. An example of a Corine Land Cover area is presented in Figure 5.5.

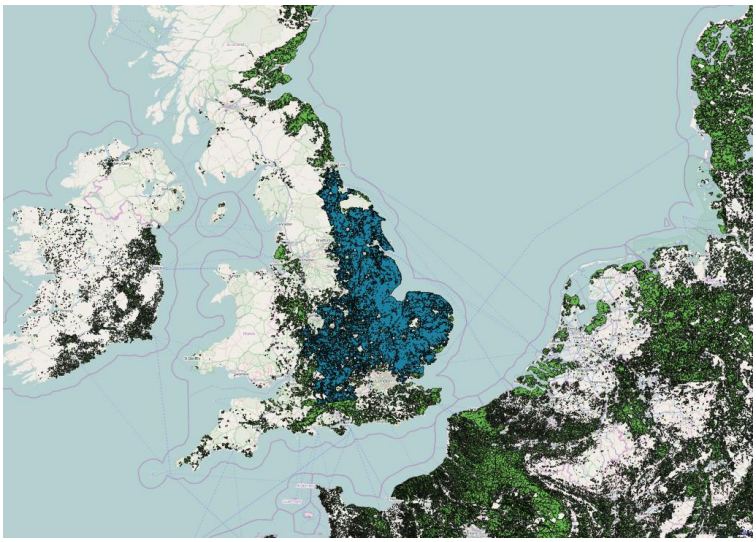

Figure 5.5: Example of a Corine Land Cover area

We developed an ontology for the the Corine Land Use/Land Cover dataset that derives from the dataset published by the European Environmental Agency[5]. We started with the ontology that was developed in TELEIOS for this dataset, but we updated it to be compliant with the GeoSPARQL vocabulary. The main classes of the developed ontology are `clc:Area`, `clc:LandUse` and `clc:Geometry` . The class `clc:LandUse` is the top class of the CLC taxonomy and contains classes that represent the land use of an area. The class `clc:Area` represents every area that has been characterized by EEA with a specific land use. The class `clc:Geometry` represents the spatial extent of an area. The main properties

[5] `http://www.eea.europa.eu/data-and-maps/data/clc-2006-vector-data-version-3`

of an instance of class `clc:Area` are `clc:hasLandUse` that represents the usage/coverage of an area, and `clc:hasGeometry` that links an area to the serialization of its geometry.

In Table 5.5 we show an example of an area that has been characterized as non-irrigated arable land.

Table 5.5: Example Area (see figure 5.5) of CLC dataset

code_00	ID	Remark	Area_Ha	Shape_Leng	Shape_Area
211	EU-2204078	NULL	3981304.06	42610763.58	39813040619.19

The RDF triples that represent the area presented in Table 5.5 are:

```
clc:EU-2204078 rdf:type clc:Area;
               clco:hasLandUse clc:NonIrrigatedArableLand ;
               clco:hascode_00 "211" ;
               clco:hasArea_Ha "3981304.06192" ;
               clco:hasShape_Leng "42610763.5868" ;
               clco:hasShape_Area "39813040619.2" ;
               geo:hasGeometry clc:Geometry_EU-2204078 .

clc:Geometry_EU-2204078
               rdf:type clc:Polygon ;
               geo:asWKT "<http://spatialreference.org/ref/epsg/etrs89/>
                          POLYGON (3578100.067199999467 3596707.2704,
                          ... ,
                          3588417.4635 3590977.5712)"^^geo:wktLiteral .
```

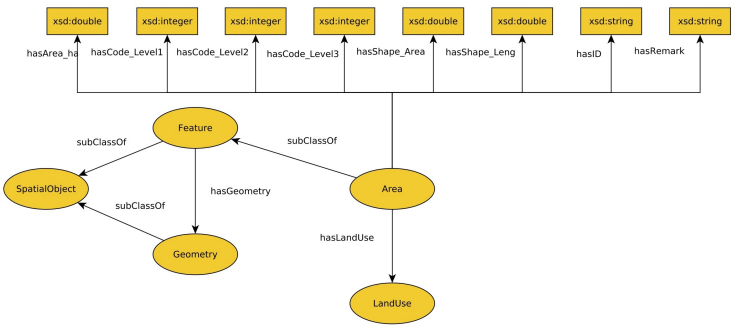

Figure 5.6: Classes in the Corine Land Cover Ontology

5.1.4 Administrative geography of Germany

The German Administrative Geography[6] is a dataset containing thematic and spatial information regarding the states of Germany. The dataset describes the administrative divisions of Germany (e.g., state, municipality). Figure 5.7 presents the states and the administrative districts of Germany.

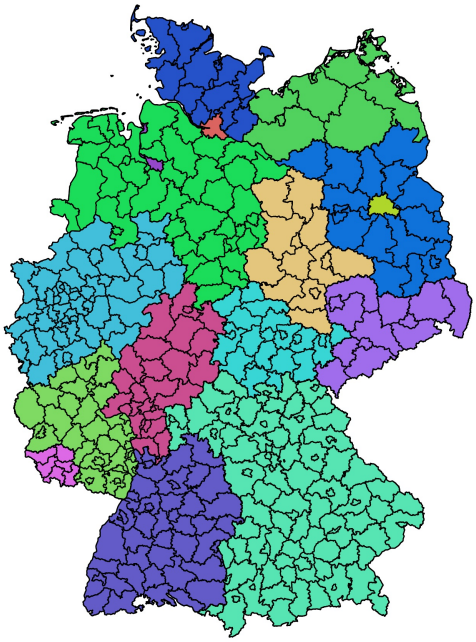

Figure 5.7: States and administrative districts of Germany

In Table 5.6 we show the information that is available for the state of Hamburg.

Table 5.6: Administative Geography of Germany data

USER	GEN	DES	ISN	WIRK-SAMKEI	DE-BKG_ID	LENGTH	SHAPE_AREA
2	Hamburg	Freie und Hanses-tadt	22	1974/01/01	DE-BKGDL20000E6GD	247768.69	739022897.08

In Figure 5.8 we present the ontology that we developed for representing the German Administrative geography. The ontology models an administrative area using three main classes: `gau:Staat`, `gau:Verwaltungseinheit`, and `gau:Bundeslander` for representing states, ad-

[6] `http://www.geodatenzentrum.de/geodaten/gdz_rahmen.gdz_div?gdz_spr=deu&gdz_akt_zeile=5&gdz_anz_zeile=1&gdz_unt_zeile=14&gdz_user_id=0`

ministrative units and federated units. The RDF triples representing an administrative unit of Germany, based on the information derived from the shapefile, are the following:

```
gau:Lnder_2 a Lnder ;
    gauo:debkg_id "DEBKGDL20000E6GD" ;
    gauo:hat_amtliche_bezeichnung "Freie und Hansestadt" ;
    gauo:hat_geographischer_name "Hamburg" ;
    gauo:hat_isn 22.0E0 ;
    gauo:hat_use 2.0E0 ;
    gauo:length 247768.69 ;
    gauo:shape_area 739022897.08 ;
    gauo:wirksamkei "1974-01-01"^^xsd:date;
    gauo:hasGeometry gau:Geometry_9402748 .

gau:Geometry_9402748 rdf:type geo:Geometry ;
    geo:asWKT "MULTIPOLYGON ((11.3931707 47.5856015,
                              ... ,
                              11.5841895 47.6494972))"^^geo:wktLiteral .
```

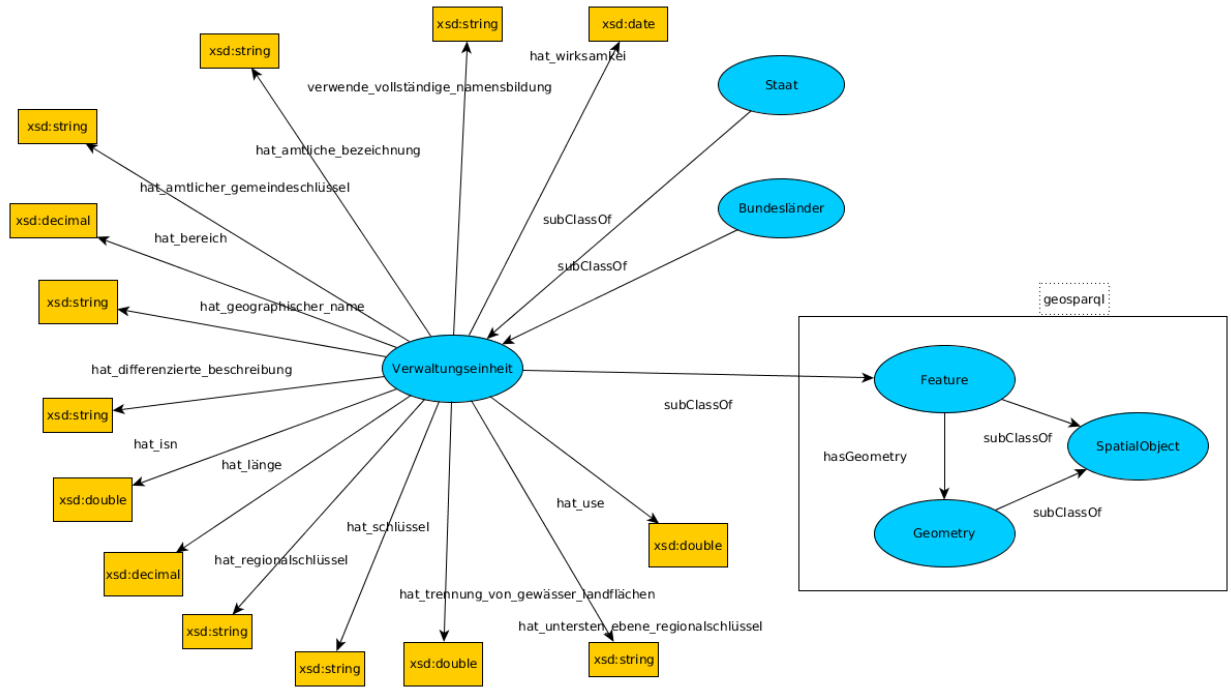

Figure 5.8: Classes in the German Administrative Units Ontology

5.1.5 LinkedGeoData

LinkedGeoData (LGD)[7] is a project focused on publishing OpenStreetMap (OSM)[8] data as linked data. OSM maintains a community-driven global editable map that gathers map data in a crowdsourcing fashion. The respective ontology is derived mainly from OSM tags, i.e., attribute-value annotations of nodes, ways, and relations. The data model of OpenStreetMap consists of three main categories: `Nodes` that represent point locations with latitude/longitude values, `Ways` that are ordered sequences of nodes, and `relations` that are groupings of nodes and ways. Figure 5.9 presents two rivers in Bavaria, Germany.

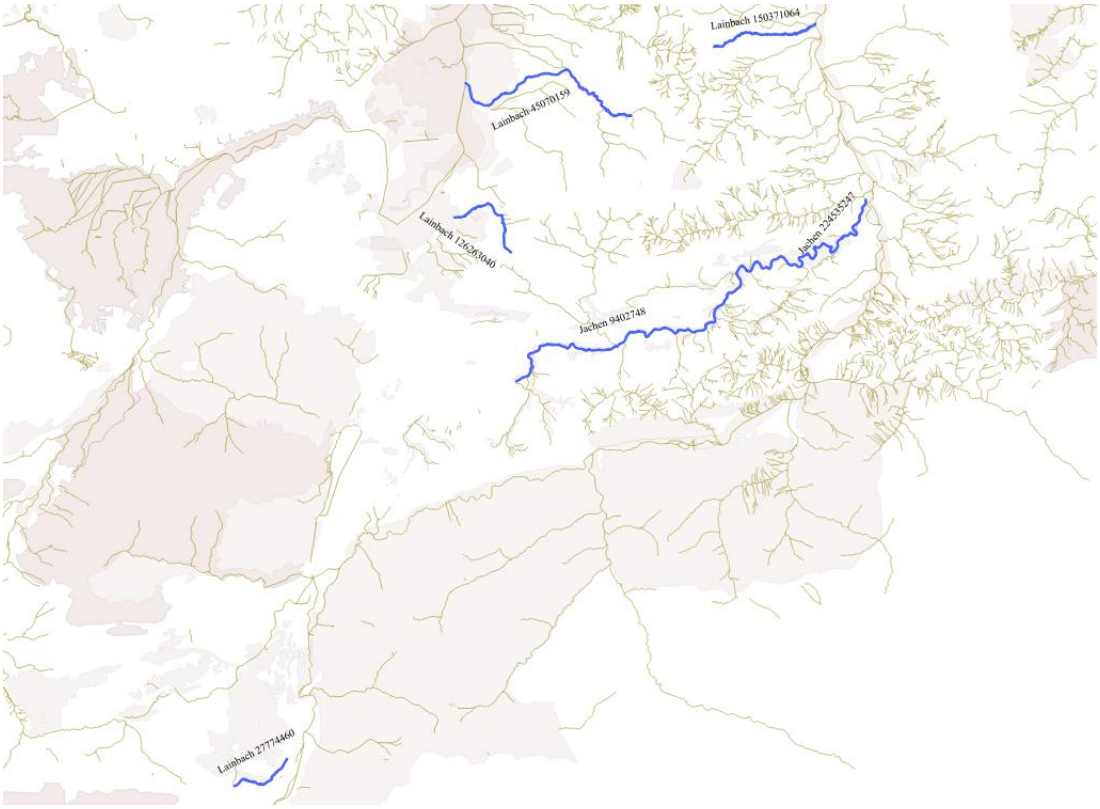

Figure 5.9: The Jachen & Lainbach rivers in Bavaria, Germany

The RDF triples that represent the rivers Jachen and Lainbach are the following:

```
osm:Waterway_9402748 rdf:type osmo:River ;
                     osmo:hasName "Jachen" ;
                     osmo:hasType "river" ;
                     osmo:hasWidth "None" ;
```

[7] http://linkedgeodata.org/
[8] http://www.openstreetmap.org/

```
                    geo:hasGeometry osm:Geometry_9402748 .
osm:Geometry_9402748 rdf:type geo:LINESTRING ;
                    geo:asWKT "LINESTRING
                              (11.3931707 47.5856015,
                               ... ,
                               11.5841895 47.6494972)"^^geo:wktLiteral .

osm:Waterway_224535247 rdf:type osmo:River ;
                       osmo:hasName "Jachen" ;
                       osmo:hasType "river" ;
                       osmo:hasWidth "None" ;
                       geo:hasGeometry osm:Geometry_224535247 .
osm:Geometry_224535247 rdf:type geo:LINESTRING ;
                       geo:asWKT "LINESTRING
                                 (11.5705614 47.6355863,
                                  ... ,
                                  11.5714161 47.6361931)"^^geo:wktLiteral .

osm:Waterway_27774460 rdf:type osmo:River ;
                      osmo:hasName "Lainbach" ;
                      osmo:hasType "river" ;
                      osmo:hasWidth "None" ;
                      geo:hasGeometry osm:Geometry_27774460 .
osm:Geometry_27774460 rdf:type geo:LINESTRING ;
                      geo:asWKT "LINESTRING
                                (11.2372667 47.4393502,
                                 ... ,
                                 11.2658696 47.4488869)"^^geo:wktLiteral .

osm:Waterway_45110595 rdf:type osmo:Stream ;
                      osmo:hasName "Lainbach" ;
                      osmo:hasType "stream" ;
                      osmo:hasWidth "None" ;
                      geo:hasGeometry osm:Geometry_45070159 .
osm:Geometry_45070159 rdf:type geo:LINESTRING ;
                      geo:asWKT "LINESTRING
                                (11.4581125 47.6821132,
```

```
                                ... ,
                                11.3687858 47.6952384)"^^geo:wktLiteral .

osm:Waterway_45070159 rdf:type osmo:Stream ;
                      osmo:hasName "Lainbach" ;
                      osmo:hasType "stream" ;
                      osmo:hasWidth "None" ;
                      geo:hasGeometry osm:Geometry_45110595 .
osm:Geometry_45110595 rdf:type geo:LINESTRING ;
                      geo:asWKT "LINESTRING
                                 (11.3917048 47.6329094,
                                 ... ,
                                 11.389525 47.6341773)"^^geo:wktLiteral .
```

5.2 Discovery Queries on data published by GeoTriples

In this section we demonstrate how to discover TalkingFields data, protected areas and rivers by using various thematic and spatial criteria. We also demonstrate how we can combine several datasets presented earlier in this chapter, in order to express more complex queries like spatial joins and distance queries.

Example 1. *Discover all information about TalkingFields products within a given polygon.*

```
SELECT  *
WHERE { ?field rdf:type tfo:Field ;
               tfo:hasFieldID ?field_id ;
               tfo:hasFieldName ?field_name ;
               tfo:hasFMArea ?field_fmarea ;
               tfo:hasUsage ?field_usage ;
               tfo:hasMainPeriod ?field_mperiod ;
               tfo:hasSubPeriod ?field_speriod ;
               tfo:hasWatering ?field_watering ;
               tfo:hasPocessed ?field_pocessed ;
               tfo:hasRasterCell ?cell .
         ?cell     geo:hasGeometry ?cell_geo .
         ?cell_geo geo:asWKT ?cell_geowkt .
         FILTER(geof:sfContains(
```

```
                "<http://spatialreference.org/ref/epsg/32632/>
                POLYGON ((677682 5346719, 677682 5346672,
                          678980 5348518, 677751 5349109,
                          677682 5346719))"^^
                <http://www.opengis.net/ont/geosparql#wktLiteral> ,
                ?cell_geowkt))}
}
```

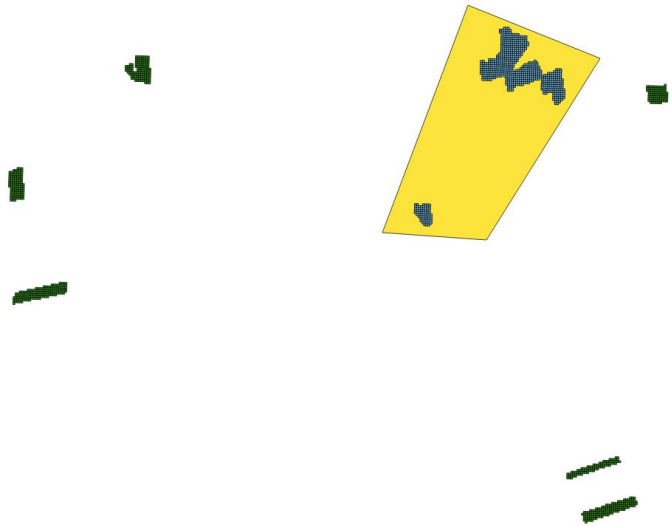

Figure 5.10: Visualization of the TalkingFields products and query region of Example 1

This query finds all information related to TalkingFields products using a spatial criterion for selecting only those fields that lie within a given polygon. Figure 5.10 visualizes the input data, the query region and the results of SPARQL query presented above. Some of the results of the query are displayed below:

?field_id	?field_name	?field_area	?field_usage	?field_mperiod	?field_speriod	?field_watering	?field_pocessed
1070	3 Mitterweg	5.101	1	2012	0	0	1
1061	7 Kreuzacker	12.629	1	2012	0	0	1
1055	11 Frankental	6.134	1	0	0	0	NULL
1068	16 Wirtsacker	7.1593	1	2013	0	0	1

Example 2. *Discover all Natura 2000 areas that lie within a given polygon.*

```
SELECT ?area_name
WHERE {?area rdf:type nat:NaturaArea ;
             nat:hasSiteName ?area_name ;
             geo:hasGeometry ?geo .
       ?geo  geo:asWKT ?geowkt .
       FILTER(geof:sfContains(
              "<http://spatialreference.org/ref/epsg/32632/>
               POLYGON ((678321 5346720, ...))"^^
               <http://www.opengis.net/ont/geosparql#wktLiteral>,
               ?geowkt))}
```

This query finds all protected areas from Natura 2000 dataset that lie within a given polygon.

Figure 5.11: Visualization of protected areas and query region of Example 2

Figure 5.11 visualizes the input data, the query region and the results of SPARQL query presented above. Some of the results of the query are displayed below:

?area_name
GEWSSERSYSTEM DER LUHE UND UNTEREN NEETZE
TRUPPENBUNGSPLTZE MUNSTER NORD UND SD
ILMENAU MIT NEBENBCHEN

Example 3. *Discover all rivers within a given polygon.*

```
SELECT  ?river_name
```

```
WHERE { ?area rdf:type osmo:River ;
              osmo:hasName ?river_name ;
              geo:hasGeometry ?geo .
              ?geo geo:asWKT ?geowkt .
        FILTER(geof:sfContains(
               "<http://www.opengis.net/def/crs/EPSG/0/4326>
               POLYGON ((123.212 323.542, ...))"^^
               <http://www.opengis.net/ont/geosparql#wktLiteral>,
               ?geowkt))}
```

Figure 5.12 visualizes the input data, the query region and the results of SPARQL query presented above. Some of the results of the query are displayed below:

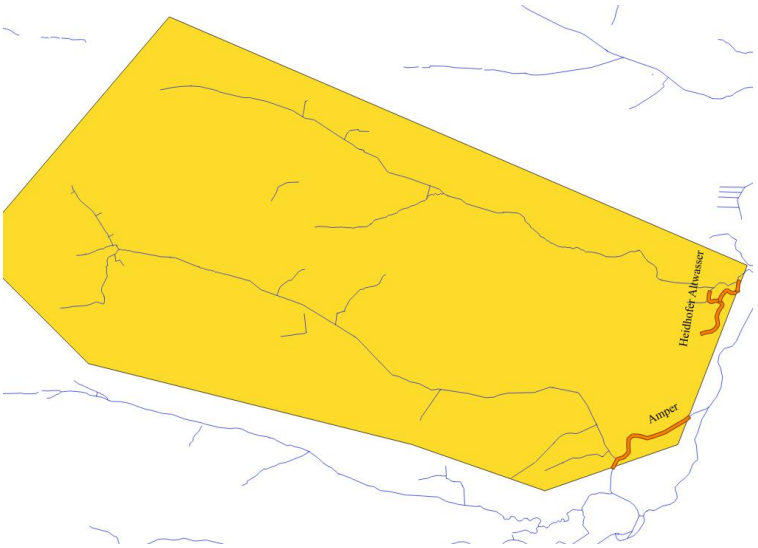

Figure 5.12: Visualization of rivers and query region of Example 3

The above SPARQL query retrieves all rivers from the Open Street Maps dataset, that lie within the given polygon. This query can be used from the precision farming application to retrieve information about rivers that are close to fields in order to produce a map for fertilizer/pesticide usage that complies to the German law. Figure 5.12 visualizes rivers from Open Street Maps, the query region and the results of SPARQL query presented above. Some of the results of the query are displayed below:

?river_name
Amper
Heidhofer Altwasser

Example 4. *Discover all information about agricultural fields that lie within a certain region of Germany.*

```
SELECT  *
WHERE { ?state rdf:type aug:Kreise ;
               gauo:hat_geographischer_name "Dachau" ;
               geo:hasGeometry ?state_geo .
        ?state_geo geo:asWKT ?state_geowkt .
        ?field rdf:type tf:Field ;
               tfo:hasFieldID ?field_id ;
               tfo:hasFieldName ?field_name ;
               tfo:field_fmarea ?field_fmarea ;
               tfo:hasUsage ?field_usage ;
               tfo:hasMainPeriod ?field_mperiod ;
               tfo:hasSubPeriod ?field_speriod ;
               tfo:hasWatering ?field_watering ;
               tfo:hasPocessed ?field_pocessed ;
               tfo:hasRasterCell ?cell .
        ?cell     geo:hasGeometry ?cell_geo .
        ?cell_geo geo:asWKT ?cell_geowkt .
        FILTER(geof:sfContains(?state_geowtk, ?cell_geowkt))
}
```

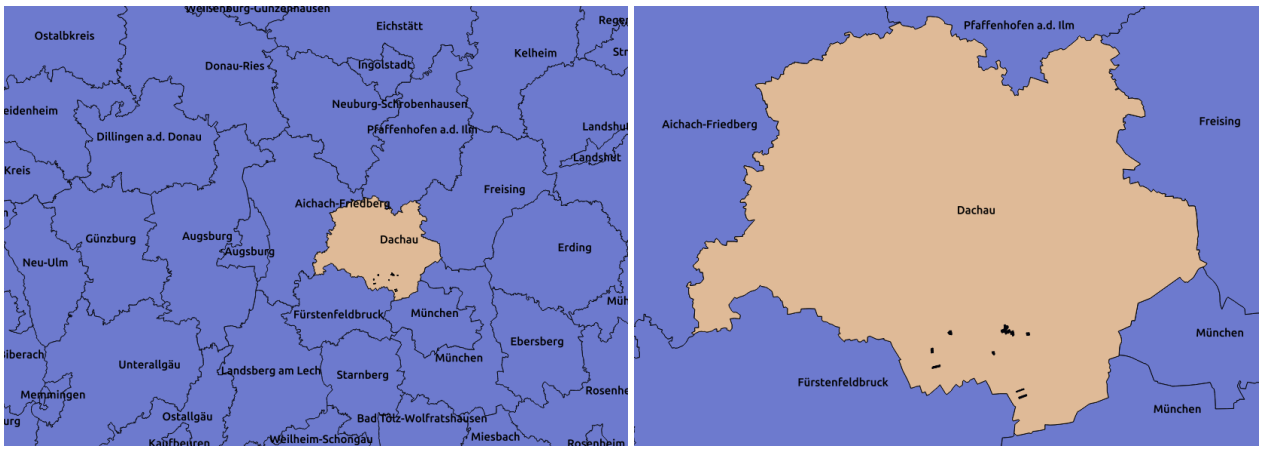

Figure 5.13: Visualisation of agricultural fields within Dachau

The above SPARQL query finds all the fields from TalkingFields dataset that lie within the district of Dachau. Notice that the query above utilizes the `geof:sfContains` SPARQL extension function in a filter clause in order to test whether the geometries of a field is contained by the geometry of the district of Dachau. Figure 5.13 visualizes the input data and the results of SPARQL query presented above. Some of the results of the query are displayed below:

?field_id	?field_name	?field_area	?field_usage	?field_mperiod	?field_sperіod	?field_watering	?field_pocessed
1070	3 Mitterweg	1	2012	0	0	1	
1061	16 Wirtsacker	1	2012	0	0	1	
1055	22 Schmidt Bauplatz	1	0	0	0	NULL	

Example 5. *Select all parts of agricultural fields that overlap a Natura 2000 area.*

```
SELECT   distinct ?field_name ?cell_x ?cell_y ?cell_cv ?cell_vigor
 WHERE { ?natarea rdf:type nato:NaturaArea ;
                  geo:hasGeometry ?natarea_geo .
         ?natarea_geo geo:asWKT ?state_geowkt .

         ?field rdf:type tf:Field ;
                tfo:hasFieldID ?field_id ;
                tfo:hasFieldName ?field_name ;
                tfo:hasUsage ?field_usage ;
                tfo:hasMainPeriod ?field_mperiod ;
                tfo:hasSubPeriod ?field_sperіod ;
                tfo:hasWatering ?field_watering ;
                tfo:hasPocessed ?field_pocessed ;
                tfo:hasRasterCell ?cell .

         ?cell  geo:hasGeometry ?cell_geo ;
                tfo:hasX ?cell_x ;
                tfo:hasY ?cell_y ;
                tfo:hasCV ?cell_cv ;
                tfo:hasVigor ?cell_vigor .
         ?cell_geo geo:asWKT ?field_geowkt .

         FILTER(geof:sfOverlaps(?natarea_geowtk, ?field_geowkt))}
```

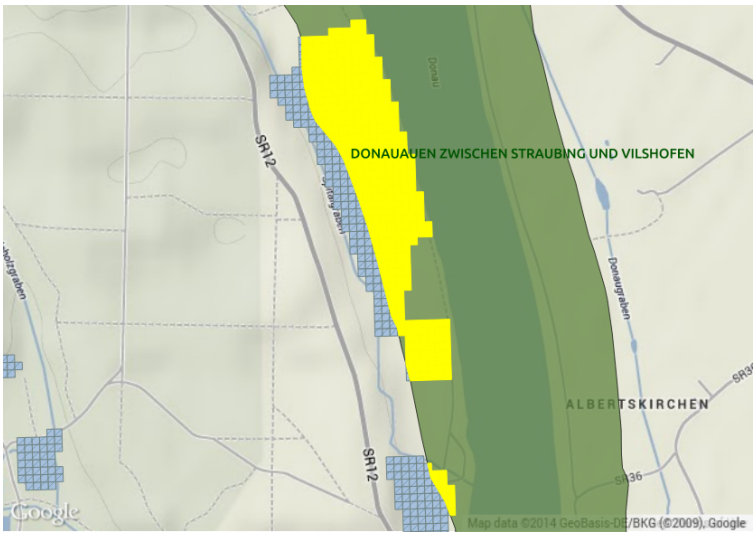

Figure 5.14: Visualisation of the parts of agricultural fields that overlap a protected area

The above SPARQL query finds all parts of a field that lie within a protected area. Similary to Example 3, the precision farming application can use this information in order to produce a map for fertilizer/pesticide usage that complies to the German law. Figure 5.14 visualizes the input data and the results of SPARQL query presented above. Some of the results of the query are displayed below:

?field_name	?cell_x	?cell_y	?cell_cv	?cell_vigor
27 Donaufeld	4534474.626	2858077.183	16.900	9999
27 Donaufeld	4534477.561	2858077.185	85.400	3
27 Donaufeld	4534473.219	2858078.151	70.300	12
50 Buchnerentau 1	4534474.627	2858233.183	74.700	5
50 Buchnerentau 1	4534474.626	2858021.228	84.800	9999
50 Buchnerentau 1	4534474.626	2858019.700	16.900	-9

Example 6. *Select all parts of agricultural fields that are less than 150 meters away from a river.*

```
SELECT  distinct ?cell_x ?cell_y ?cell_cv ?cell_vigor
 WHERE {?river rdf:type osmo:River ;
               osmo:hasName ?river_name ;
               geo:hasGeometry ?river_geo .
        ?river_geo geo:asWKT ?river_geowkt .
```

```
?field rdf:type tfo:Field ;
       tfo:hasFieldID ?field_id ;
       tfo:hasUsage ?field_usage ;
       tfo:hasMainPeriod ?field_mperiod ;
       tfo:hasSubPeriod ?field_speriod ;
       tfo:hasWatering ?field_watering ;
       tfo:hasPocessed ?field_pocessed ;
       tfo:hasRasterCell ?cell .

?cell geo:hasGeometry ?cell_geo ;
      tfo:hasX ?cell_x ;
      tfo:hasY ?cell_y ;
      tfo:hasCV ?cell_cv ;
      tfo:hasVigor ?cell_vigor .
?cell_geo geo:asWKT ?field_geowkt .

FILTER(geof:distance(?river_geowkt , ?field_geowkt,
       <http://www.opengis.net/def/uom/OGC/1.0/metre>) < 150)}
```

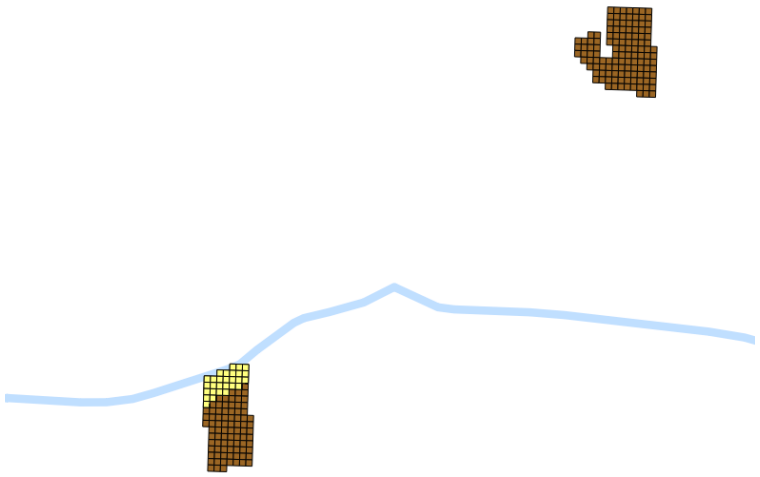

Figure 5.15: Visualisation of the parts of an agricultural field that is close to a river

This query uses the `geo:distance` function that involves two spatial objects and a URI that identifies a unit of measurement. The result of this metric function is the minimum distance between the spatial objects that are given as an input. Similarly to Example 3 and 5,

the precision farming application can use this information in order to produce an improved map for fertilizer/pesticide usage by taking into account existing information about the surroundings of a field.

Figure 5.15 visualizes the input data and the results of SPARQL query presented above. Some of the results of the query are displayed below:

?field_name	?cell_x	?cell_y	?cell_cv	?cell_vigor
19 Maisachacker gro	1858714.796	8887945.407	38.200	9999
19 Maisachacker gro	1858714.251	8887945.096	64.700	-13
19 Maisachacker gro	1858714.157	8887942.180	70.300	-10
19 Maisachacker gro	1858714.459	3652143.188	NULL	22

5.3 Summary

In this chapter we presented how GeoTriples can be used in a real world scenario. We demonstrated how to use GeoTriples to build a precision farming application that consumes linked open geospatial data. We briefly described the ontologies that we developed for representing involved data and we presented datasets from the linked open data cloud that are being used by applications such as the fire monitoring application implemented in TELEIOS. Finally, we provided some example queries that demonstrate how semantic web technologies and linked data can be used for data discovery and data manipulation in the context of a precision farming application.

Chapter 6
Conclusions

In this thesis we provided a description and a comparison among several languages for representing mappings from relation databases to RDF datasets, based on several potentially desired features. We present in more detail the language R2RML that we chose to extend for expressing mappings of EO and geospatial data to RDF. We presented the architecture of the transformation tool GeoTriples that is able to generate automatically R2RML mappings for Earth Observation data stored in spatially-enabled databases or ESRI shape files and process such mappings in order to produce RDF graphs from the input data. Then, we demonstrated how GeoTriples can be used in a real world scenario where Earth Observation products are published as linked open Earth Observation data. Following the publication step, EO data are combined with data that are currently published as linked open geospatial data in order to realize a precision farming application that offers management decision support for plant protection and fertilization activities regarding temporal and legal restrictions by taking into account information about the surrounding of the fields.

Acronyms

EO	Earth Observation
RDF	Resource Description Framework
OSM	OpenStreetMap
ESA	European Space Agency
NASA	National Aeronautics and Space Administration
RDB	Relational Database
OGC	Open Geospatial Consortium
WKT	Well-Known Text

namespace	**prefix**
`http://strdf.di.uoa.gr/ontology#`	`strdf`
`http://www.opengis.net/ont/geosparql#`	`geo`
`http://www.opengis.net/def/function/geosparql/`	`geof`
`http://www.w3.org/ns/r2rml#`	`rr`
`http://www.w3.org/1999/02/22-rdf-syntax-ns#`	`rdf`
`http://www.w3.org/2000/01/rdf-schema#`	`rdfs`
`http://www.w3.org/2002/07/owl#`	`owl`
`http://data.linkedeodata.eu/osm/ontology#`	`osmo`
`http://data.linkedeodata.eu/osm/`	`osm`
`http://data.linkedeodata.eu/Natura2000DE/ontology#`	`nato`
`http://data.linkedeodata.eu/Natura2000DE/`	`nat`
`http://data.linkedeodata.eu/corine-land-cover/ontology#`	`clco`
`http://data.linkedeodata.eu/corine-land-cover/`	`clc`
`http://data.linkedeodata.eu/talking-fields/ontology#`	`tfo`
`http://data.linkedeodata.eu/talking-fields/`	`tf`
`http://data.linkedeodata.eu/AdministrativeUnitsGermany/ontology#`	`gauo`
`http://data.linkedeodata.eu/AdministrativeUnitsGermany/`	`gau`

APPENDIX I

In this chapter, we provide installation guidelines for GeoTriples. For the convenience of the reader, we give specific instructions for executing each step in a computer running Ubuntu 14.04 64-bit. We will present how the user can install GeoTriples using a debian package, and how to compile GeoTriples from sources.

Installing GeoTriples using the debian package

In this section we will describe how one can install the GeoTriples utilities using the provided debian package and commands entered in the command-line interface terminal. The main dependency for GeoTriples is Java 1.7.

The first step is to install Java 7. More information can be found at `https://help.ubuntu.com/community/Java`.

```
# sudo apt-get install openjdk-7-jre
```

The next step is to download the respective .deb file from the software repository. The debian package file can be downloaded from the following URL: `http://sourceforge.net/projects/geotriples/files/geotriples_1.0.deb/download`

After downloading the debian package, the user should install it using some package manager like dpkg.

```
# sudo dpkg -i packageFile
```

Finally, GeoTriples can be invoked by typing either of the two following commands:

```
# geotriples-cmd [ARGS]
```

which invokes the command line utility and

```
# geotriples-gui
```

which initializes the graphical user interface of GeoTriples.

In Appendix II we present how the user can use the command line utility and the graphical user interface of GeoTriples respectively.

Building GeoTriples from sources

In this section we provide detailed instructions on how to compile GeoTriples from sources. Before downloading and compiling GeoTriples, the following steps should be executed:

- Install Java 7. More information can be found at `https://help.ubuntu.com/community/Java`.

  ```
  # sudo apt-get install openjdk-7-jdk
  ```

- Install maven. More information can be found at `http://maven.apache.org/`

  ```
  # sudo apt-get install maven
  ```

- Install Git. More information can be found at `http://git-scm.com/`.

  ```
  # sudo apt-get install git
  ```

- Install the 32-bit shared libraries of the GNU C library for AMD64.

  ```
  # sudo apt-get install libc6:i386
  ```

The first step is to download the source code of GeoTriples via Git:

```
# git clone https://github.com/LinkedEOData/GeoTriples.git
```

The above command creates a replica of the source code of GeoTriples in a directory called geotriples-code. Now the user can proceed with the compilation of the downloaded sources to create the necessary executables. The source code can be compiled by executing:

```
# cd geotriples-code && mvn clean package
```

After compiling the source code, two executable all-in-one jars are created inside the target directory. These are the `target/geotriples-VERSION-SNAPSHOT-cmd.one-jar.jar` and
`target/geotriples-VERSION-SNAPSHOT-gui.one-jar.jar` which implement the command line and graphical utilities respectively. These executables can be run by executing the following command:

```
# java -jar target/geotriples-{VERSION}-SNAPSHOT-{UTILITY}.one-jar.jar
```

where `VERSION` is the current version of GeoTriples (1.0) and `UTILITY` can be either `cmd` or `gui`.

APPENDIX II

Using the command line interface of GeoTriples

In this chapter we will present in detail how the user can use the command line interface of GeoTriples for publishing EO data as an RDF graph. The procedure for transforming EO data into RDF comprises three steps: the automatic generation of an R2RML mapping given an input data set, the optional step of editing the R2RML mapping by the user in order to follow the ontology of her preference, and finally the utilization of the R2RML mapping for the automatic generation of the RDF graph.

Automatic generation of R2RML mappings

The *mapping generator* component of GeoTriples is responsible for the automatic generation of an R2RML mapping document by analyzing the schema of the input data source, that can be either an existing database or an ESRI Shapefile. This mapping document associates each table with a new RDFS class identified by a URI that is generated automatically according to the name of the table, and each column to a property with a URI that is generated automatically from the name of the column. This mapping document can be used as-is or can be further customized by the user in order to satisfy her needs (e.g., comply with a different vocabulary). The command for generating such a mapping is:

```
# geotriples generate-mapping [options] jdbcURLorFileURL
```

or

```
# geotriples generate-mapping [-u user] [-p password] [-d driver]
                              [-b baseURI] [-o r2rmloutfile]
                              [-r2rml] jdbcURLorFileURL
```

The parameters `user`, `password` and `driver` are used when the input data source is a spatially-enabled relational database and allow the specification of the credentials to be used when connecting to the database. If the parameter `r2rmloutfile` is defined, then the generated mapping will be redirected to the specified file. The argument `r2rml` can be specified

in order generate a mapping document according to the R2RML mapping language. The parameter `baseURI` is used for turning relative URI and URI patterns into absolute URI. Finally, the argument `jdbcURLorFileURL` is JDBC url for connecting to MonetDB, PostgreSQL, Oracle, MySQL, SQL Server, HSQLDB or Interbase/Firebird database. Currently, GeoTriples supports the automatic generation of mappings for columns containing geometric information for the MonetDB and PostgreSQL databases.

Publishing EO data as an RDF graph

The *R2RML processor* of GeoTriples allows the user to publish the contents of a spatially-enabled database or the contents of an ESRI Shapefile as an RDF graph, given an R2RML mapping document. The R2RML processor of GeoTriples can be invoked as follows:

```
# geotriples dump-rdf [options] inputmappingfile
```

or

```
# geotriples dump-rdf [-u user] [-p password] [-d driver] [-jdbc jdbcURL]
                      [-sh fileURL] [-f format] [-b baseURI] [-o rdfoutfile]
                      inputmappingfile
```

Using the `format` argument, the user specifies the desired RDF syntax for the generated RDF graph. GeoTriples supports TURTLE, RDF/XML, RDF/XML-ABBREV, N3, and N-TRIPLES which is the default syntax. The parameter `rdfoutfile` redirects the generated RDF triples into the specified file. Finally, the argument `inputmappingfile` defines the R2RML document that specifies how the input data source is mapped to an RDF graph.

Using GeoTriples for transforming database into RDF

In this section* we show how GeoTriples can be used to publish a dataset containing geospatial information using the command line interface. In this example we show how the user can publish data produced by the talkingfields project[1] when stored in MonetDB. The input dataset contains information about fields in Bavaria, Germany and follows the schema depicted in Figure 6.1. Some sample data from each table is depicted in Table 6.1.

[1] `http://www.talkingfields.de`

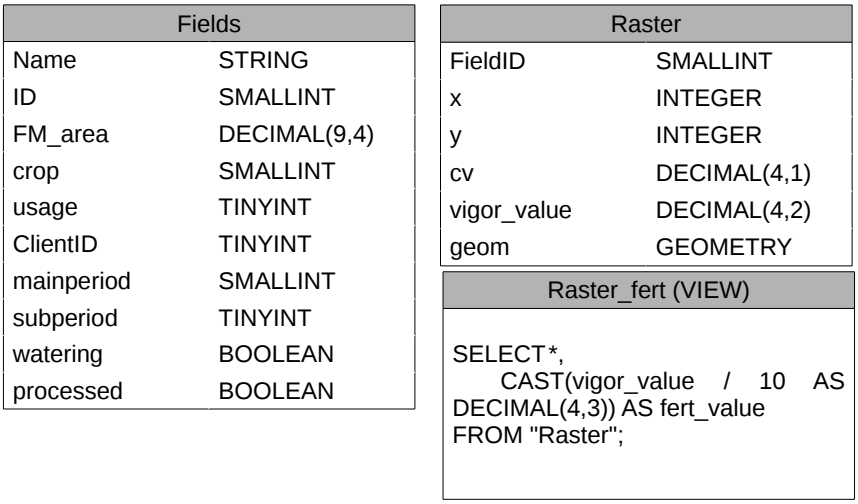

Figure 6.1: Database schema for a talkingfields product.

Name	ID	FM_area	crop	usage	ClientID
2 Keller re.	1049	6.8426	101	1	34
4 Wischlburg li.	1067	10.3118	null	1	34

mainperiod	subperiod	watering	processed
2013	0	false	true
2013	0	false	true

FieldID	x	y	cv	vigor_value	geom
1034	334760	5410460	8.2	null	POLYGON ((334760.00 5410460.00,...))
1048	335580	5411940	94.2	-7.00	POLYGON ((335580.00 5411940.00,...))

Table 6.1: Sample data of a talkingfields product.

The first step is to generate the R2RML mapping document using the `generate-mapping` module of GeoTriples. Assuming that the user has installed GeoTriples using the provided .deb package, the R2RML document is generated with the following command:

```
# geotriples generate-mapping -u monetdb -p monetdb \
                              -b http://data.linkedeodata.eu/talking-fields \
                              -o mapping.ttl -r2rml \
                              jdbc:monetdb://server:50000/tf
```

This command produces an R2RML mapping document with name `mapping.ttl` for the data stored in the database `tf` that is served by MonetDB that runs on a machine with name `server` on port `50000`. We now present parts of the generated R2RML mapping document.

```
@prefix geof: <http://www.opengis.net/def/function/geosparql/>.
@prefix map: <#>.
@prefix ogc: <http://www.opengis.net/ont/geosparql#>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix rr: <http://www.w3.org/ns/r2rml#>.
@prefix rrx: <http://www.w3.org/ns/r2rml-ext#>.
@prefix rrxf: <http://www.w3.org/ns/r2rml-ext/functions/def/>.
@prefix vocab: <ontology#>.
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.

map:rs_Fields
    rr:logicalTable [ rr:tableName '"tf"."rs"."Fields"'; ];
    rr:subjectMap [ rr:class vocab:rs_Fields;
      rr:template \newline
'http://data.linkedeodata.eu/talking-fields/Fields/id/{"ID"}'; ];
    rr:predicateObjectMap [
        rr:predicate vocab:rs_Fields_ClientID;
        rr:objectMap [
            rr:datatype xsd:integer;
            rr:column '"ClientID"';
        ];
    ];
    rr:predicateObjectMap [
        rr:predicate vocab:rs_Fields_mainperiod;
        rr:objectMap [
            rr:datatype xsd:integer;
            rr:column '"mainperiod"';
        ];
    ];
    .
map:rs_RasterGeo
    rr:logicalTable [ rr:sqlQuery """SELECT gid, st_dimension(geom) as
    dimension, st_dimension(geom) as "coordinateDimension",
    st_dimension(geom) as "spatialDimension", st_issimple(geom) as
    "isSimple", st_isempty(geom) as "isEmpty",
    CONCAT('<http://www.opengis.net/def/crs/EPSG/0/32633> ' ,
```

```
    REPLACE(CAST(geom AS TEXT), '"', '')) as
"asWKT" FROM "tf"."rs"."Raster\""""; ];
    rr:subjectMap [ rr:class ogc:Geometry;
      rr:template \newline
'http://data.linkedeodata.eu/talking-fields/RasterGeo/Geometry/{"gid"}'; ];
    rr:predicateObjectMap [
        rr:predicate ogc:dimension;
        rr:objectMap [
            rr:datatype xsd:integer;
            rr:column '"dimension"';
        ];
    ];
    rr:predicateObjectMap [
        rr:predicate ogc:asWKT;
        rr:objectMap [
            rr:datatype ogc:wktLiteral;
            rr:column '"asWKT"';
        ];
    ];
    rr:predicateObjectMap [
        rr:predicate ogc:isSimple;
        rr:objectMap [
            rr:datatype xsd:boolean;
            rr:column '"isSimple"';
        ];
    ];
    .
map:rs_Raster
    rr:logicalTable [ rr:tableName '"tf"."rs"."Raster"'; ];
    rr:subjectMap [ rr:class vocab:rs_Raster;
      rr:template
'http://data.linkedeodata.eu/talking-fields/Raster/id/{"gid"}'; ];
    rr:predicateObjectMap [
        rr:predicate vocab:rs_Raster_cv;
        rr:objectMap [
            rr:datatype xsd:decimal;
            rr:column '"cv"';
```

```
        ];
    ];
    rr:predicateObjectMap [
        rr:predicate ogc:hasGeometry;
        rr:objectMap [
            rr:parentTriplesMap map:rs_RasterGeo;
            rr:joinCondition [ rr:child "gid"; rr:parent "gid"; ];
        ];
    ];
    rr:predicateObjectMap [
        rr:predicate vocab:rs_Raster_FieldID;
        rr:objectMap [
            rr:parentTriplesMap map:rs_Fields;
            rr:joinCondition [ rr:child '"FieldID"'; rr:parent '"ID"'; ];
        ];
    ];
    .
map:rs_Raster_fert
    rr:logicalTable [ rr:tableName '"tf"."rs"."Raster_fert"'; ];
    rr:subjectMap [ rr:class vocab:rs_Raster_fert;
      rr:template
  'http://data.linkedeodata.eu/talking-fields/Raster_fert/id/{"gid"}'; ];
    rr:predicateObjectMap [
        rr:predicate vocab:rs_Raster_fert_fert_value;
        rr:objectMap [
            rr:datatype xsd:decimal;
            rr:column '"fert_value"';
        ];
    ];
    .
```

Optionally, the user may edit the contents of the R2RML mapping document (e.g. in order to utilize a different vocabulary). In LEO, we developed an ontology for talkingfields products that was presented in Deliverable D5.2.1. Since we want the generated RDF graph to follow this vocabulary, we modify accordingly the generating R2RML mapping to follow the developed vocabulary. We now present parts of the modified R2RML mapping document.

```
@prefix geof: <http://www.opengis.net/def/function/geosparql/>.
@prefix map: <#>.
@prefix ogc: <http://www.opengis.net/ont/geosparql#>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix rr: <http://www.w3.org/ns/r2rml#>.
@prefix rrx: <http://www.w3.org/ns/r2rml-ext#>.
@prefix rrxf: <http://www.w3.org/ns/r2rml-ext/functions/def/>.
@prefix vocab: <ontology#>.
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.

map:rs_Fields
    rr:logicalTable [ rr:tableName '"tf"."rs"."Fields"'; ];
    rr:subjectMap [ rr:class vocab:Field;
     rr:template
'http://data.linkedeodata.eu/talking-fields/000001/Field/id/{"ID"}'; ];
    rr:predicateObjectMap [
        rr:predicate vocab:hasClientID;
        rr:objectMap [
            rr:datatype xsd:integer;
            rr:column '"ClientID"';
        ];
    ];
    rr:predicateObjectMap [
        rr:predicate vocab:hasMainPeriod;
        rr:objectMap [
            rr:datatype xsd:integer;
            rr:column '"mainperiod"';
        ];
    ];
    .
map:rs_RasterGeo
    rr:logicalTable [ rr:sqlQuery """SELECT gid, st_dimension(geom) as
    dimension, st_dimension(geom) as "coordinateDimension",
    st_dimension(geom) as "spatialDimension", st_issimple(geom) as
    "isSimple", st_isempty(geom) as "isEmpty",
    CONCAT('<http://www.opengis.net/def/crs/EPSG/0/32633> ' ,
```

```
    REPLACE(CAST(geom AS TEXT), '"', '')) as "asWKT" FROM
    "tf"."rs"."Raster\""""; ];
    rr:subjectMap [ rr:class ogc:Geometry;
     rr:template
'http://data.linkedeodata.eu/talking-fields/000001/Geometry/id/{"gid"}'; ];
    rr:predicateObjectMap [
        rr:predicate ogc:dimension;
        rr:objectMap [
            rr:datatype xsd:integer;
            rr:column '"dimension"';
        ];
    ];
    rr:predicateObjectMap [
        rr:predicate ogc:asWKT;
        rr:objectMap [
            rr:datatype ogc:wktLiteral;
            rr:column '"asWKT"';
        ];
    ];
    rr:predicateObjectMap [
        rr:predicate ogc:isSimple;
        rr:objectMap [
            rr:datatype xsd:boolean;
            rr:column '"isSimple"';
        ];
    ];
    .
map:rs_Raster
    rr:logicalTable [ rr:sqlQuery """SELECT ROW_NUMBER() OVER
    (ORDER BY "x", "y") as "gid", "x", "y", "vigor_value",
    "cv", "FieldID", CAST("vigor_value" / 10 AS DECIMAL(4,3))
    AS "fert_value" FROM "tf"."rs"."RasterFert\""""; ];
    rr:subjectMap [ rr:class vocab:RasterCell;
     rr:template
 'http://data.linkedeodata.eu/talking-fields/000001/RasterCell/id/{"gid"}'; ];
    rr:predicateObjectMap [
        rr:predicate vocab:hasFertValue;
```

```
        rr:objectMap [
            rr:datatype xsd:decimal;
            rr:column '"fert_value"';
        ];
    ];
    rr:predicateObjectMap [
        rr:predicate vocab:hasCV;
        rr:objectMap [
            rr:datatype xsd:decimal;
            rr:column '"cv"';
        ];
    ];
    rr:predicateObjectMap [
        rr:predicate ogc:hasGeometry;
        rr:objectMap [
            rr:parentTriplesMap map:rs_RasterGeo;
            rr:joinCondition [ rr:child '"gid"'; rr:parent '"gid"'; ];
        ];
    ];
    rr:predicateObjectMap [
        rr:predicate vocab:belongsToField;
        rr:objectMap [
          rr:template
     'http://data.linkedeodata.eu/talking-fields/000001/Field/id/{"FieldID"}';
        ];
    ];
    .
```

Next, we use the R2RML processor of GeoTriples for generating an RDF graph for the talkingfields products stored in the database `tf`:

```
# geotriples dump-rdf -u monetdb -p monetdb \
                      -b http://linkedeodata.eu/talking-fields \
                      -o tf.nt
                      -jdbc jdbc:monetdb://server:50000/tf \
                      mapping-leo.ttl
```

This command produces an RDF document with the name `tf.nt`. Part of the contents of the generated RDF graph is presented below.

```
<http://data.linkedeodata.eu/talking-fields/000001/Field/id/1034> a tf:Field ;
      tf:hasClientID "34"^^xsd:integer> ;
      tf:hasSubPeriod "0"^^xsd:integer ;
      tf:isProcessed "true"^^xsd:boolean ;
      tf:hasWatering "false"^^xsd:boolean ;
      tf:hasName "1 Gei\u00DFelweiher"xsd:string ;
      tf:hasUsage "1"^^xsd:integer ;
      tf:hasMainPeriod "2013"^^xsd:integer ;
      tf:hasFM_Area "27.3071"^^xsd:decimal ;
      rdfs:label "Field #1034" .


<http://data.linkedeodata.eu/talking-fields/000001/RasterCell/id/14718>
      a tf:RasterCell ;
      tf:hasX "334760"^^xsd:integer ;
      tf:hasY "5410460"^^xsd:integer ;
      tf:hasCV "8.2"^^xsd:decimal ;
      tf:hasFertValue "0"^^xsd:decimal ;
      tf:hasVigor> "0"^^xsd:decimal ;
      tf:belongsToField
<http://data.linkedeodata.eu/talking-fields/000001/Field/id/1034> ;
      ogc:hasGeometry
<http://data.linkedeodata.eu/talking-fields/000001/Geometry/id/14718> .

<http://data.linkedeodata.eu/talking-fields/000001/Geometry/id/14718>
      a ogc:Geometry ;
      rdfs:label "demobetriebGeo #1" ;
      ogc:asWKT "<http://www.opengis.net/def/crs/EPSG/0/32633>POLYGON
      ((334760 5410460, 334760 5410440,...))"^^ogc:wktLiteral ;
      ogc:coordinateDimension 2 ;
      ogc:dimension 2 ;
      ogc:isEmpty "false"^^xsd:boolean ;
      ogc:isSimple "true"^^xsd:boolean ;
      ogc:spatialDimension 2 .
```

Using the graphical user interface of GeoTriples

In addition to the command line version of GeoTriples, we implemented a graphical user interface that allows the users to interact with GeoTriples through graphical icons and visual indicators such as ontology-driven forms. The graphical user interface uses the Apache Pivot[2] framework, an open-source platform for building installable web and desktop applications with a rich variety of widgets.

In this chapter we present the structure of the graphical user interface and repeat the previous example using the graphical user interface instead of the command line utility of GeoTriples.

Structure of the graphical user interface of GeoTriples

The graphical user interface of GeoTriples, presented in Figure 6.2, consists of the following parts: the menu bar, the mapping editor, the mapping viewer and the tool bar.

The menu bar provides easy access to the main functionalities of GeoTriples. By choosing the option `Connect` from the submenu `File`, the user is prompted to input the details for the input data source as shown in Figure 6.4. The user may choose a relational database or an ESRI shape file as an input data source. Afterwards, the tool accesses the data source defined by the user and read its schema. For instance, if the user wants to use a relational database as an input data source, she will be asked to input the necessary information database engine and name, username and password as depicted in Figure 6.5. Via the menu `Publish`, the user can access the two core functionalities of GeoTriples; `Generate R2RML Mapping` and `Generate RDF graph`.

The mapping editor of the graphical user interface of GeoTriples allows the user to customize an R2RML mapping. The mapping editor is initially populated according to the schema of the input data source. The mapping editor allows the user to customize the R2RML mapping that will be generated while the mapping viewer displays to the user the generated R2RML document in Turtle notation.

The mapping editor supports the extensions of the R2RML mapping language that we defined in section 4.4.The mapping viewer is updated whenever the user clicks the `Generate R2RML Mapping` option from the submenu `Publish`.

The `Toolbox` section contains buttons that can be used as shortcuts for the `Generate`

[2] `https://pivot.apache.org/`

`R2RML Mapping` and `Generate RDF graph` operations. Additionally, this component provides the end user with the ability to select the desired syntax for the generated RDF graph as well as the coordinate reference system that will be used for the generated spatial RDF literals by entering the appropriate EPSG code.

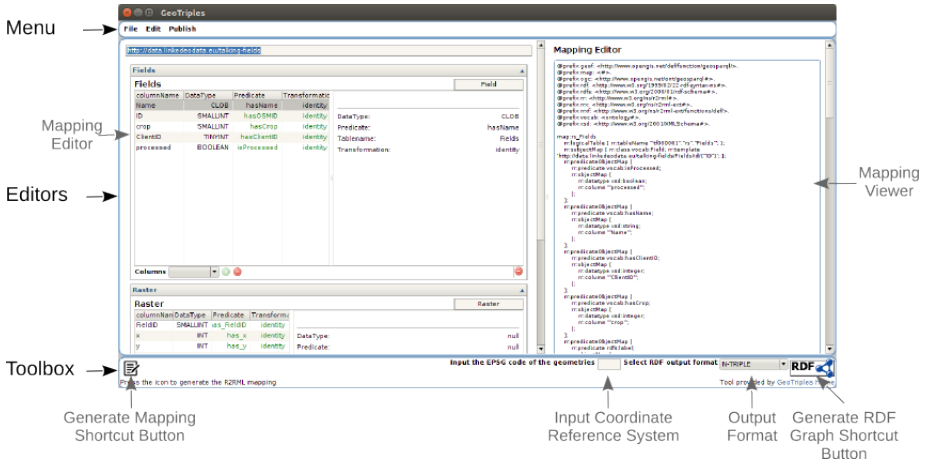

Figure 6.2: Part of the GeoTriples graphical user interface

Using GeoTriples for transforming database into RDF

In this section we will show how GeoTriples is used to publish a dataset containing geospatial information. Specifically, we will use the graphical user interface of the application to create first of all the R2RML mapping from RDB schema to RDF and then use the produced mapping to create an RDF dump from the data source. We will use the same MonetDB database (*tf*) as in Section 6 with schema presented in Figure 6.1.

In this example we will map to RDF all tables from the database. For the table Fields we will map to RDF only the columns `Name`, `ID`, `crop`, `ClientID`. We will also make use of the talkingfields ontology that was developed in LEO and map the aforementioned columns to an RDF graph using the properties of the ontology.

Let us now present how the user can use the graphical user interface of GeoTriples for generating an RDF graph for a talkingfields product that is stored in a spatially-enabled MonetDB database. The first step is to establish a connection to the database using the `File` → `Connect` option as shown in Figure 6.3.

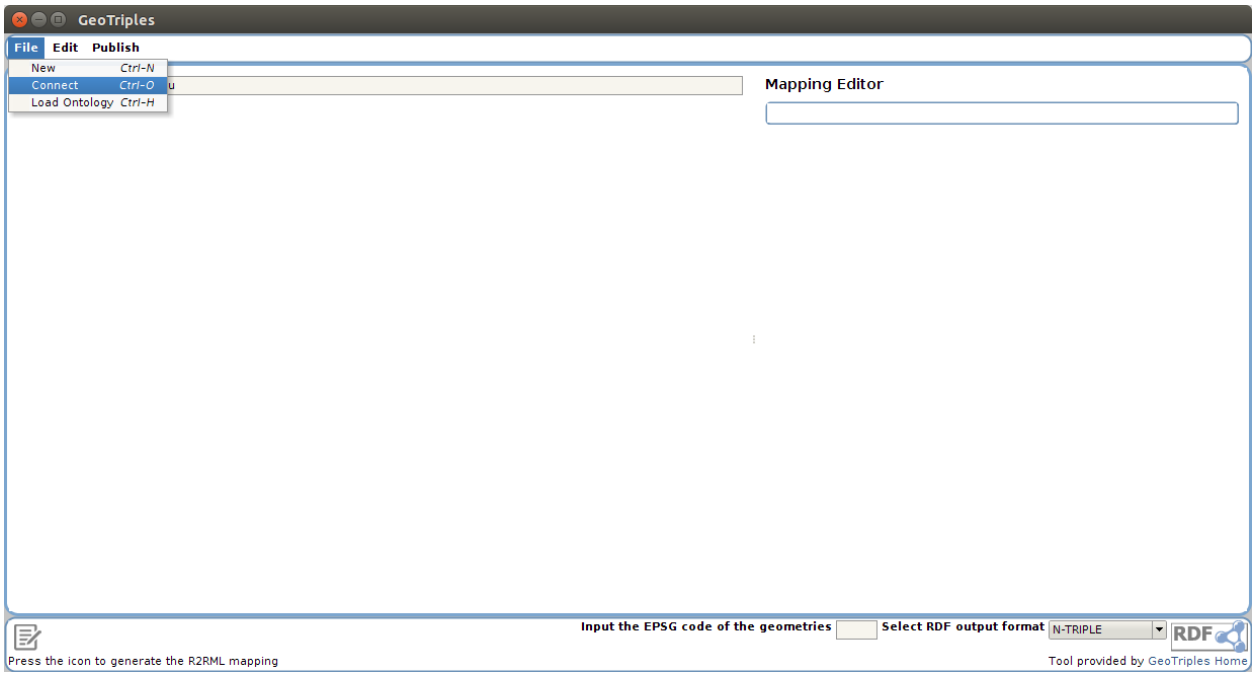

Figure 6.3: Menu File → Connect

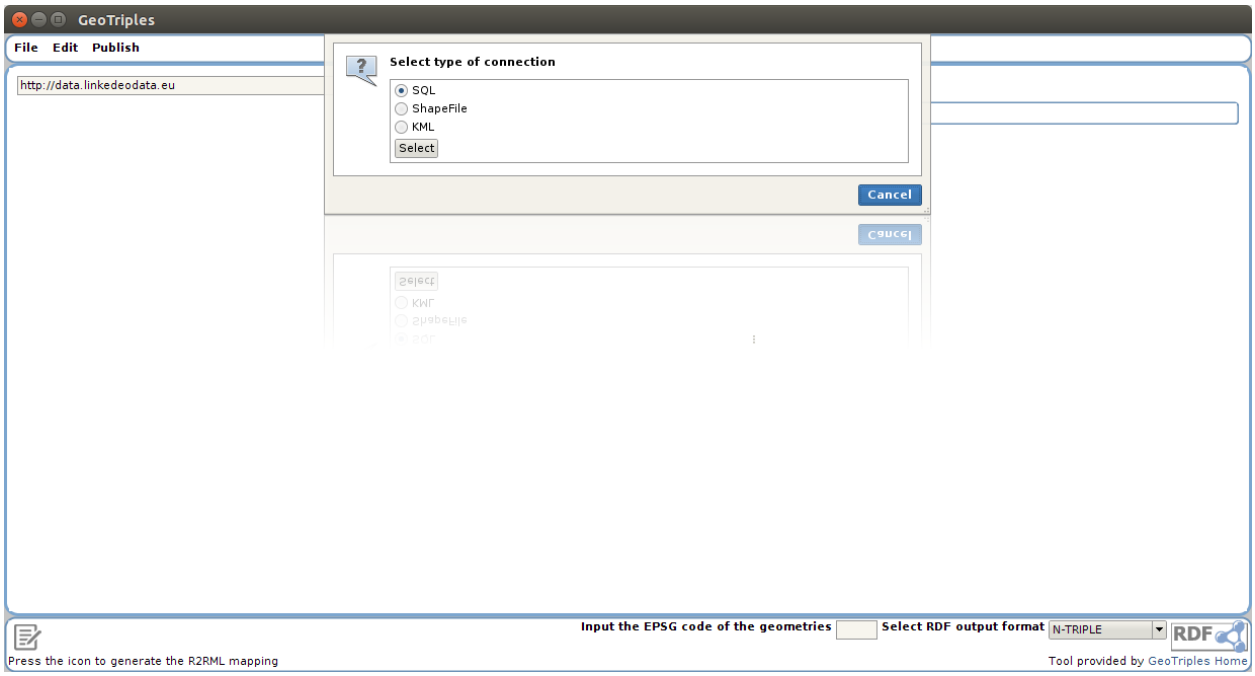

Figure 6.4: Select SQL type of connection

In the pop up form that opens, the user fills in all required information for connecting to the database as shown in Figure 6.5.

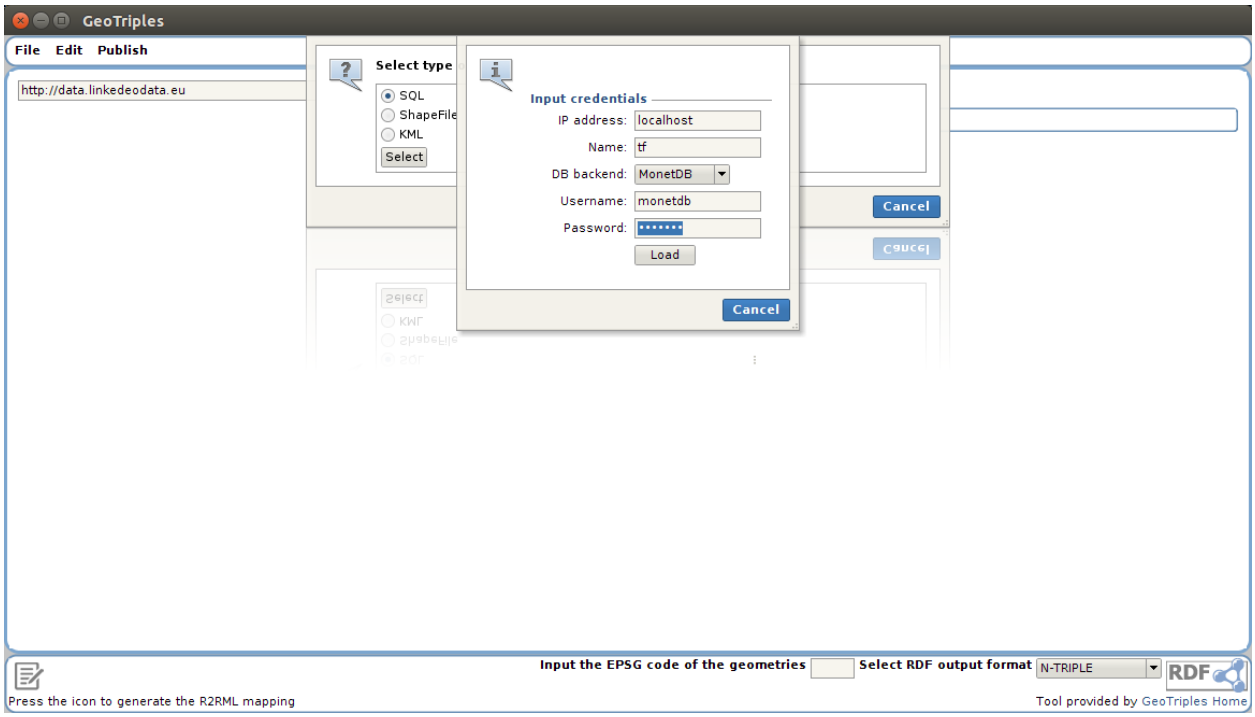

Figure 6.5: Provide connection properties

After connecting to the database, GeoTriples reads the schema of the input database and populates the contents of the mapping editor. Then, the user may edit the mapping according to her needs. For example, the user may remove tables and columns that should not be published by selecting them and clicking the remove button under the table description.

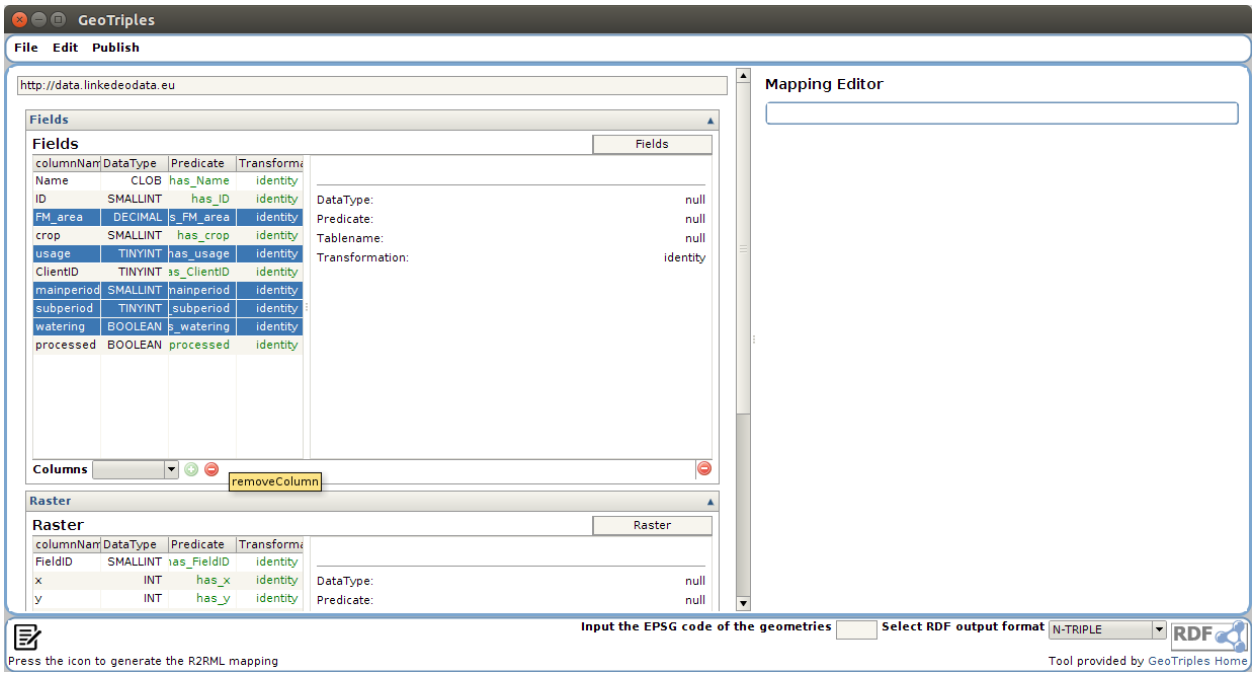

Figure 6.6: Select columns to remove

The user can also modify the R2RML mapping to follow a different vocabulary from the one that is been generated automatically by GeoTriples. In this example, we want to generate an RDF graph that follows the talkingfields ontology. For loading the developed ontology, the user should choose menu `File` → `Load Ontology` as shown in Figure 6.7. After loading the ontology, the mapping editor is automatically updated, thus allowing the user to make use of the classes and properties that are defined in the chosen ontology as shown in Figure 6.8,6.9.

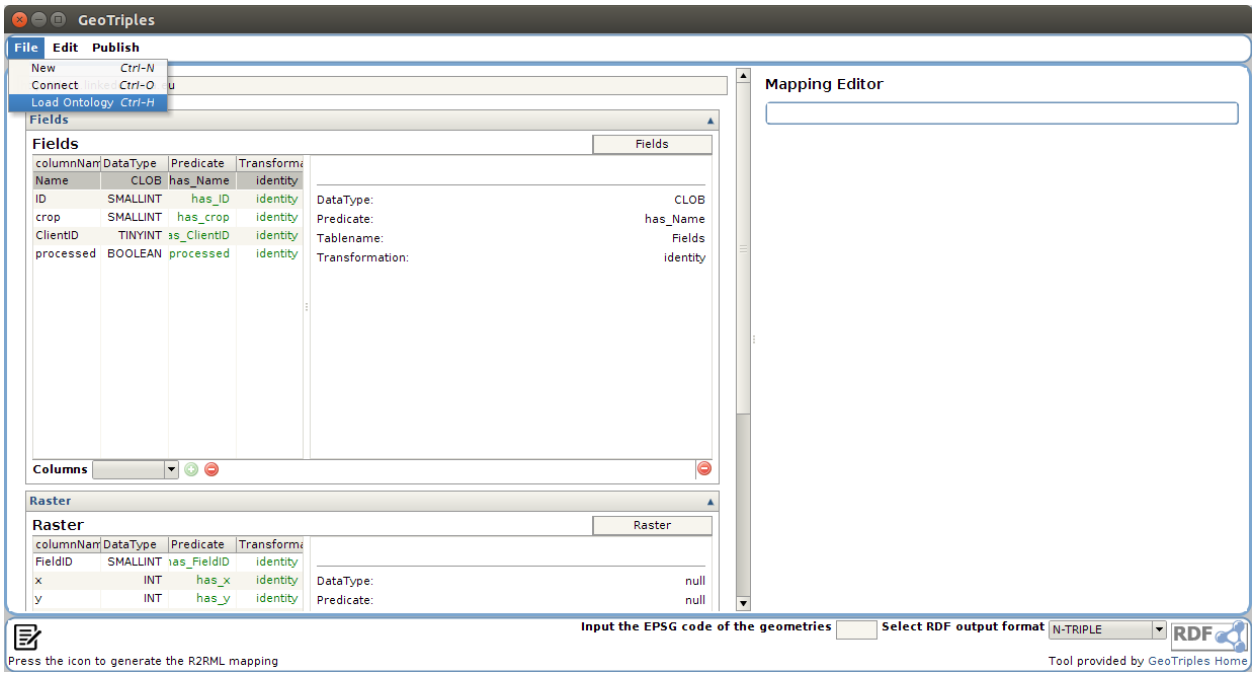

Figure 6.7: Load an existing ontology

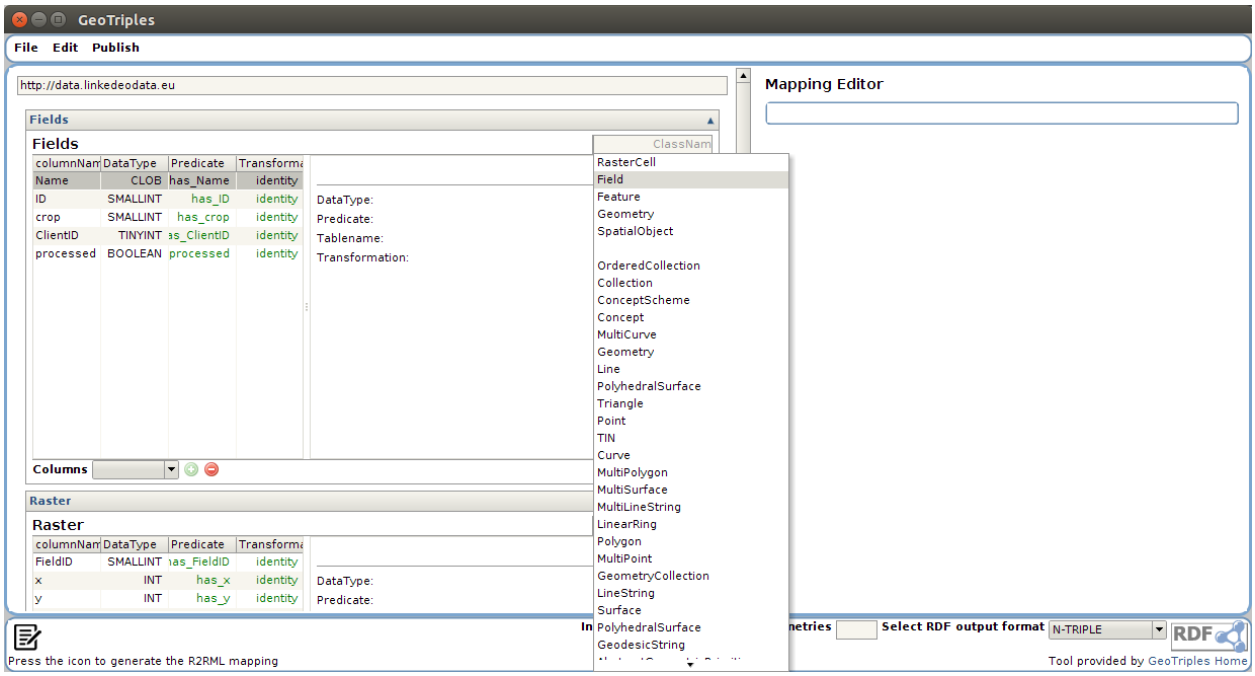

Figure 6.8: Change class of table Fields using the classes from loaded ontology

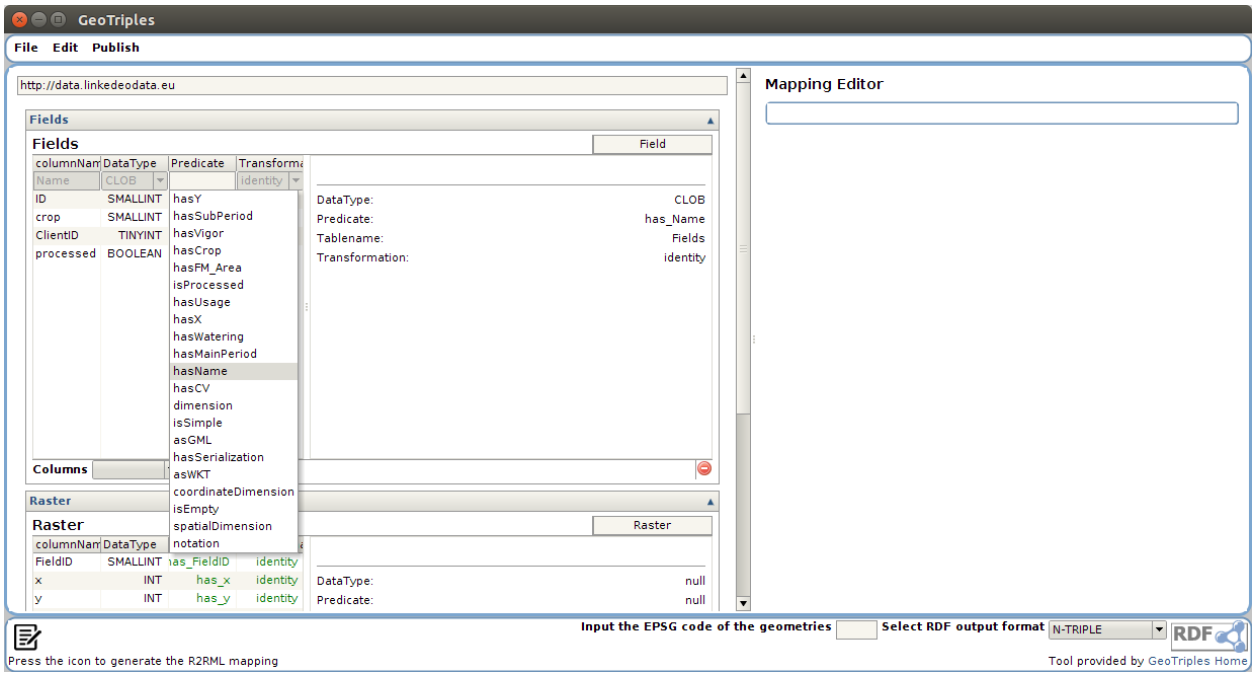

Figure 6.9: Change predicate of column using the properties from loaded ontology

The last thing to consider, before generating an RDF graph, is to define the baseIRI of the generated R2RML document. This parameter is used for converting relative URI and URI patterns to absolute URI. Figure 6.10 depicts how a user can modify the baseIRI parameter.

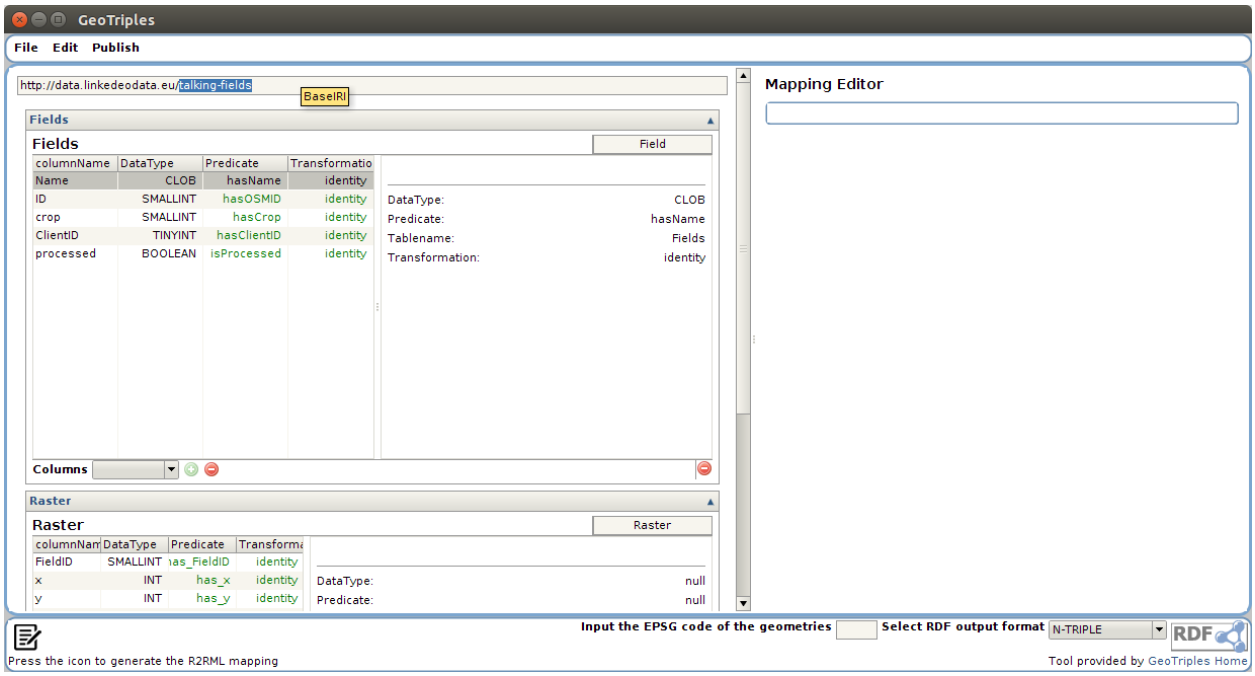

Figure 6.10: Provide the baseIRI

After customizing the R2RML mappings using the mapping editor, the user clicks `Publish` → `Generate R2RML Mapping` in order to generate the final R2RML mapping that will be used afterwards for the generation of the RDF graph. The user may inspect the generated R2RML mapping as shown in Figure 6.12.

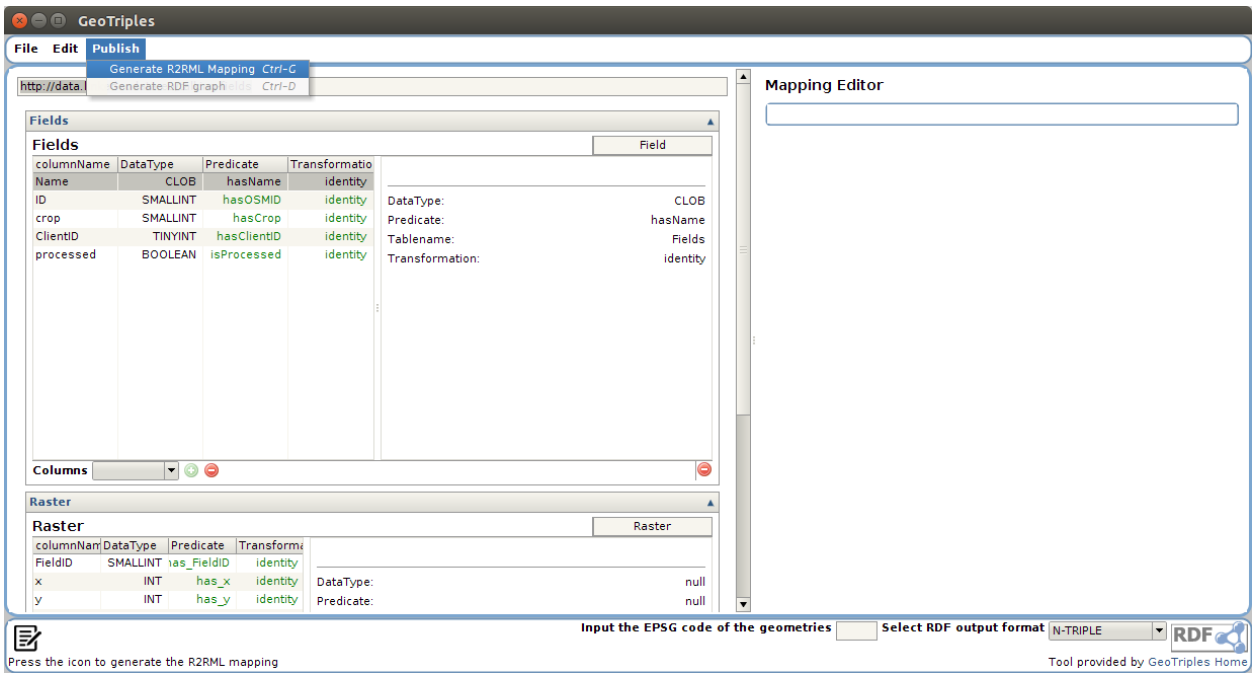

Figure 6.11: Menu Publish → Generate R2RML Mapping using information from Mapping Editor

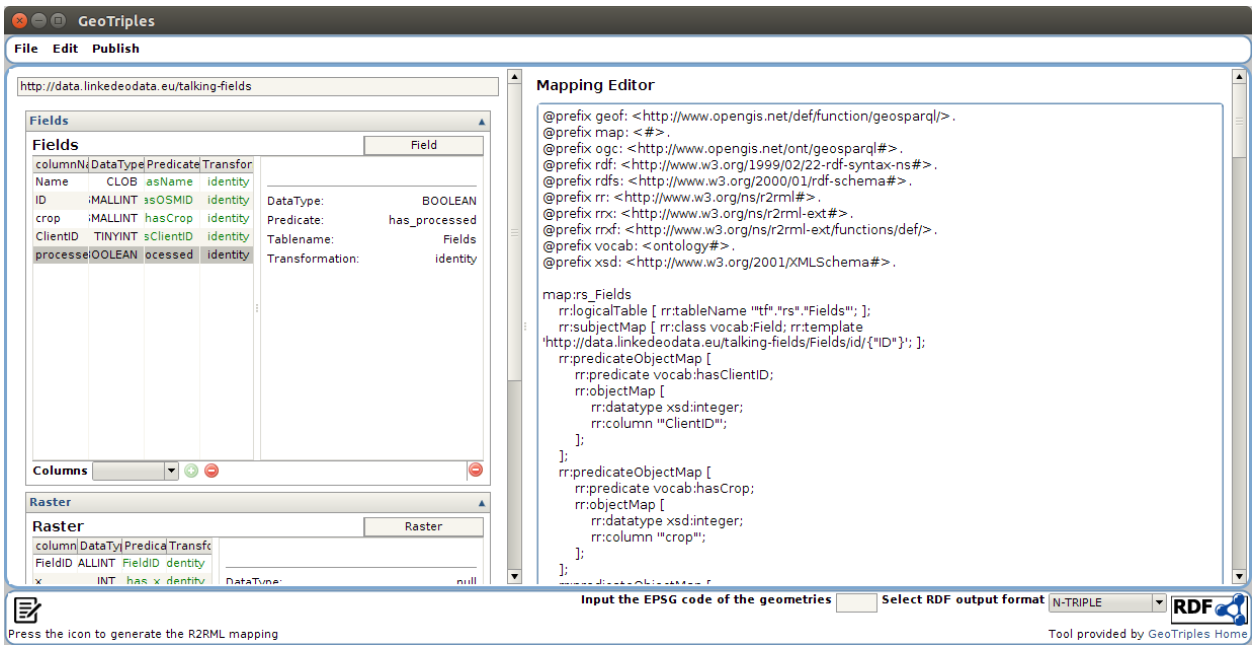

Figure 6.12: Preview or Edit R2RML mapping

The last step is to generate the RDF graph using the modified R2RML mapping document that was produced in the previous steps. This step can be done by selecting `Publish` → `Generate RDF graph` as shown in Figure 6.13.

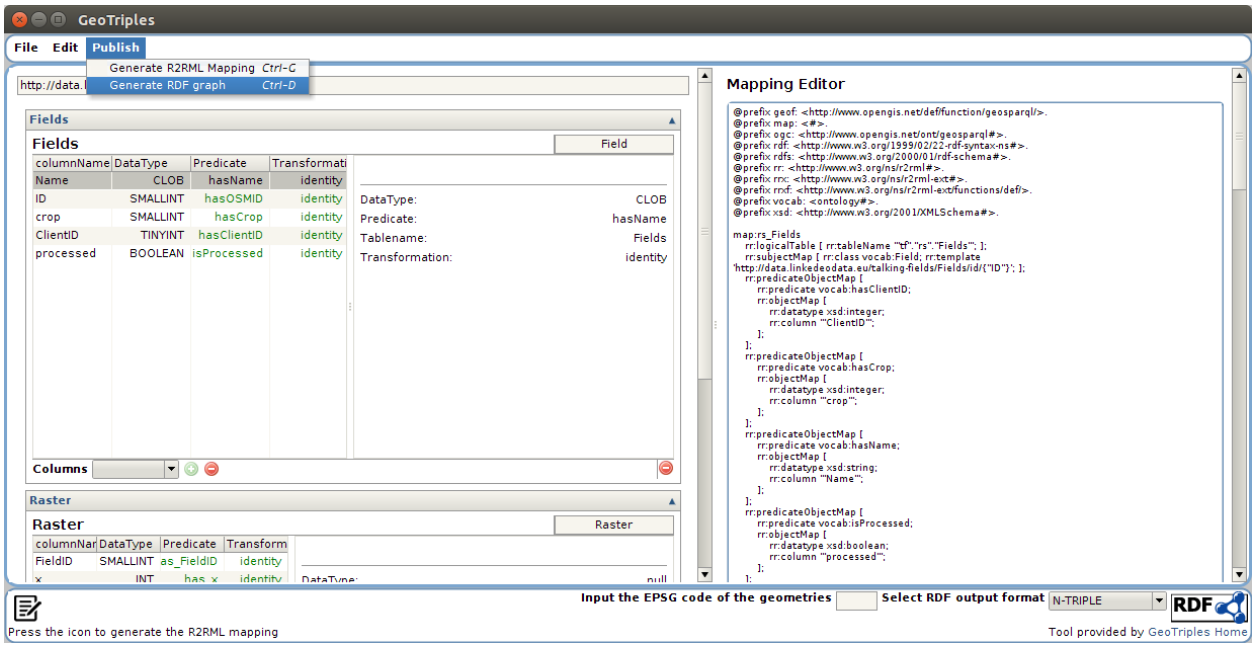

Figure 6.13: Menu Publish → Generate RDF graph

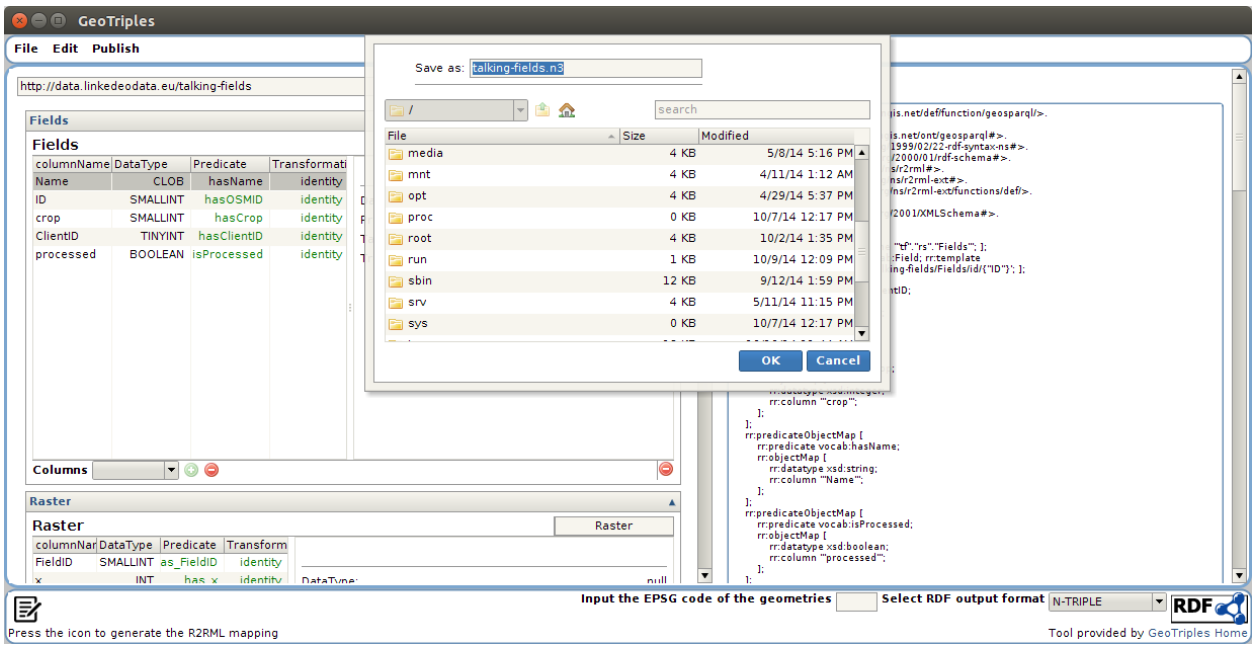

Figure 6.14: Save result in a file

Bibliography

[1] SquirrelRDF. http://jena.sourceforge.net/SquirrelRDF/.

[2] Open Geospatial Consortium. OpenGIS Catalogue Services Specification. OpenGIS Implementation Specification, 2007. Available from: `http://portal.opengeospatial.org/files/?artifact_id=20555`.

[3] Open Geospatial Consortium. OpenGIS Implementation Specification for Geographic information - Simple feature access - Part 2: SQL option. OpenGIS Implementation Standard, 2010. Available from: `http://portal.opengeospatial.org/files/?artifact_id=25354`.

[4] Open Geospatial Consortium. OpenGIS Implementation Standard for Geographic information - Simple f eature access - Part 1: Common Architecture. OpenGIS Implementation Standard, 2010. Available from: `http://portal.opengeospatial.org/files/?artifact_id=25355`.

[5] F. Bancilhon and N. Spyratos. update semantics of relational views.

[6] Jesus Barrasa, Oscar Corcho, and Ascun Gómez-Pérez. R2O, an Extensible and Semantically based Database-to-Ontology Mapping Language. In *in In Proceedings of the 2nd Workshop on Semantic Web and Databases (SWDB2004)*, pages 1069–1070. Springer, 2004.

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

[8] Christian Bizer. D2r map - a database to rdf mapping language. In Irwin King and Tamás Máray, editors, *WWW (Posters)*, 2003.

[9] Christian Bizer, Tom Heath, and Tim Berners-Lee. Linked data - the story so far. *International Journal on Semantic Web and Information Systems*, 5(3):1–22, 2009.

[10] Christian Bizer and Andy Seaborne. D2RQ-treating non-RDF databases as virtual RDF graphs. In *Proceedings of the 3rd international semantic web conference (ISWC2004)*, volume 2004, 2004.

[11] Stefan Burgstaller and Wolfgang Angermair. Ontologies and linked data for precision farming-v1. Del. D5.2.1, FP7 project LEO, 2014.

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

[13] Richard Cyganiak, Chris Bizer, Jrg Garbers, Oliver Maresch, and Christian Becker. The D2RQ Platform. http://d2rq.org/.

[14] Souripriya Das, Seema Sundara, and Richard Cyganiak. R2RML: RDB to RDF Mapping Language. W3C Recommendation. http://www.w3.org/TR/r2rml/.

[15] Cristian Pérez de Laborda and Stefan Conrad. Relational.OWL: A Data and Schema Representation Format Based on OWL. In *Proceedings of the 2Nd Asia-Pacific Conference on Conceptual Modelling - Volume 43*, APCCM '05, pages 89–96. Australian Computer Society, Inc., 2005.

[16] Orri Erling and Ivan Mikhailov. RDF Support in the Virtuoso DBMS. In Sören Auer, Christian Bizer, Claudia Müller, and Anna V. Zhdanova, editors, *CSSW*, volume 113 of *LNI*, pages 59–68. GI, 2007.

[17] Matthias Hert. Relational databases as semantic web endpoints. In *The Semantic Web: Research and Applications*, volume 5554 of *Lecture Notes in Computer Science*, pages 929–933. Springer Berlin Heidelberg, 2009.

[18] Matthias Hert, Gerald Reif, and Harald C. Gall. Updating Relational Data via SPARQL/Update. In *Proceedings of the 2010 EDBT/ICDT Workshops*, EDBT '10, pages 24:1–24:8, New York, NY, USA, 2010. ACM.

[19] Matthias Hert, Gerald Reif, and Harald C. Gall. A comparison of rdb-to-rdf mapping languages. In *Proceedings of the 7th International Conference on Semantic Systems*, I-Semantics '11, pages 25–32, New York, NY, USA, 2011. ACM.

[20] Manolis Koubarakis. Linked Open Earth Observation Data: The LEO Project. ESA-EUSC-JRC 2014, 2014.

[21] Manolis Koubarakis, Manos Karpathiotakis, Kostis Kyzirakos, Charalampos Nikolaou, and Michael Sioutis. Data Models and Query Languages for Linked Geospatial Data. In Thomas Eiter and Thomas Krennwallner, editors, *Reasoning Web. Semantic Technologies for Advanced Query Answering*, volume 7487 of *Lecture Notes in Computer Science*, pages 290–328. Springer Berlin / Heidelberg, 2012.

[22] Manolis Koubarakis and Kostis Kyzirakos. Modeling and querying metadata in the semantic sensor web: The model stRDF and the query language stSPARQL. In *ESWC*, 2010.

[23] Kostis Kyzirakos, Manos Karpathiotakis, and Manolis Koubarakis. Strabon: A Semantic Geospatial DBMS. In *In Proceedings of the 11th International Semantic Web Conference*, Lecture Notes in Computer Science, 2012.

[24] OGC. GeoSPARQL - A geographic query language for RDF data, November 2010.

[25] Kostas Patroumpas, Michalis Alexakis, Giorgos Giannopoulos, and Spiros 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.

[26] Freddy Priyatna, Óscar Corcho, and Juan Sequeda. Formalisation and experiences of R2RML-based SPARQL to SQL query translation using Morph. In Chin-Wan Chung, Andrei Z. Broder, Kyuseok Shim, and Torsten Suel, editors, *WWW*, pages 479–490. ACM, 2014.

[27] O. Software. Mapping Relational Data to RDF with Virtuoso's RDF Views. http://virtuoso.openlinksw.com/whitepapers/relationalg.html.

[28] Richard Cyganiak Souripriya Das, Seema Sundara. R2RML: RDB to RDF Mapping Language, September 2012.