Abstract: This document presents two tools, Sparqlify and TripleGeo, for transforming geospatial data from several conventional formats, into RDF triples, compliant with several standards (GeoSPARQL, Vistuoso vocabulary, etc). Also, it discusses the state of art in geospatial and RDF knowledge extraction and possible workflows for updating changes from source datasets in destination datasets.

The information in this document reflects only the author’s views and the European Community is not liable for any use that may be made of the information contained therein. The information in this document is provided “as is” without guarantee or warranty of any kind, express or implied, including but not limited to the fitness of the information for a particular purpose. The user thereof uses the information at his/her sole risk and liability.
History

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Reason</th>
<th>Revised by</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>02/06/2013</td>
<td>First version and section outline</td>
<td>Kostas Patroumpas</td>
</tr>
<tr>
<td>0.1</td>
<td>09/06/2013</td>
<td>First draft on chapter 3 describing TripleGeo</td>
<td>Kostas Patroumpas</td>
</tr>
<tr>
<td>0.2</td>
<td>09/06/2013</td>
<td>First draft on chapter 2 describing Sparqlify</td>
<td>Claus Stadler</td>
</tr>
<tr>
<td>0.3</td>
<td>10/06/2013</td>
<td>First integrated deliverable</td>
<td>Giorgos Giannopoulos</td>
</tr>
<tr>
<td>0.4</td>
<td>13/06/2013</td>
<td>Addition of Introduction chapter</td>
<td>Kostas Patroumpas</td>
</tr>
<tr>
<td>0.5</td>
<td>19/06/2013</td>
<td>Version prepared for the project’s peer-review</td>
<td>Kostas Patroumpas, Claus Stadler</td>
</tr>
<tr>
<td>0.6</td>
<td>25/06/2013</td>
<td>Improvements based on internal review comments</td>
<td>Kostas Patroumpas, Spiros Athanasiou</td>
</tr>
<tr>
<td>0.7</td>
<td>26/06/2013</td>
<td>Additions and improvements based on internal review comments</td>
<td>Claus Stadler, Giorgos Giannopoulos</td>
</tr>
<tr>
<td>0.8</td>
<td>22/08/2013</td>
<td>Final Chapter 1 additions</td>
<td>Claus Stadler</td>
</tr>
<tr>
<td>1.0</td>
<td>23/08/2013</td>
<td>Final version</td>
<td>Kostas Patroumpas</td>
</tr>
</tbody>
</table>

Author List

<table>
<thead>
<tr>
<th>Organisation</th>
<th>Name</th>
<th>Contact Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATHENA</td>
<td>Spiros Athanasiou</td>
<td><a href="mailto:spathan@imis.athena-innovation.gr">spathan@imis.athena-innovation.gr</a></td>
</tr>
<tr>
<td>ATHENA</td>
<td>Lily Bezati</td>
<td><a href="mailto:lbez@imis.athena-innovation.gr">lbez@imis.athena-innovation.gr</a></td>
</tr>
<tr>
<td>ATHENA</td>
<td>Giorgos Giannopoulos</td>
<td><a href="mailto:giann@imis.athena-innovation.gr">giann@imis.athena-innovation.gr</a></td>
</tr>
<tr>
<td>ATHENA</td>
<td>Kostas Patroumpas</td>
<td><a href="mailto:kpatro@dblab.ece.ntua.gr">kpatro@dblab.ece.ntua.gr</a></td>
</tr>
<tr>
<td>ATHENA</td>
<td>Dimitris Skoutas</td>
<td><a href="mailto:dskoutas@imis.athena-innovation.gr">dskoutas@imis.athena-innovation.gr</a></td>
</tr>
<tr>
<td>InfAI</td>
<td>Claus Stadler</td>
<td><a href="mailto:cstadler@informatik.uni-leipzig.de">cstadler@informatik.uni-leipzig.de</a></td>
</tr>
</tbody>
</table>

Time Schedule before Delivery

<table>
<thead>
<tr>
<th>Next Action</th>
<th>Deadline</th>
<th>Care of</th>
</tr>
</thead>
<tbody>
<tr>
<td>First version</td>
<td>09/06/2013</td>
<td>Kostas Patroumpas (ATHENA), Claus Stadler (InfAI)</td>
</tr>
<tr>
<td>First consolidated version</td>
<td>19/06/2013</td>
<td>Kostas Patroumpas (ATHENA), Claus Stadler (InfAI)</td>
</tr>
<tr>
<td>Peer review</td>
<td>25/06/2013</td>
<td>Hugh Williams (OpenLink)</td>
</tr>
<tr>
<td>Submit final restructured version</td>
<td>23/08/2013</td>
<td>Kostas Patroumpas (ATHENA)</td>
</tr>
</tbody>
</table>
Executive Summary

This document provides a short background on knowledge extraction and Extract-Transform-Load processes on geospatial and RDF data, discusses workflows for updating destination datasets based on changes on source datasets, and presents two implemented prototypes, that fill the gap in the area of transforming conventional geospatial data into RDF data. Specifically, we describe:

- *Sparqlify*, a SPARQL-SQL rewriter that enables one to define RDF views on relational databases and query them with SPARQL, providing, thus, access to billions of virtual triples from the OpenStreetMap database, through the LinkedGeoData Server.
- *TripleGeo*, a geospatial ETL tool that can take as input standard geospatial data (ESRI shapefiles, spatial tables in Oracle Spatial, PostGIS, MySQL and IBM DB2 databases), and transform them into RDF triples in several RDF formats and CRS, while maintaining compliance with the GeoSparql standard, Virtuoso custom vocabulary, or WGS84 RDF Geoposition vocabulary.

The layout of the document is the following.

In Chapter 1, we introduce background knowledge on knowledge extraction and transformation of geospatial and RDF data. We also discuss possible workflows for dealing with updates on the initial dataset and their propagation to the produced RDF datasets, justifying the implementations of the tools presented in the following chapters.

In Chapter 2, we present *Sparqlify*, a SPARQL-to-SQL rewriter which enables SPARQL queries on relational databases, emphasizing on the LinkedGeoData framework which utilizes *Sparqlify* to provide access to OpenStreetMap data in RDF form, through SPARQL endpoints and downloadable data dumps. We describe in detail the main components of the framework, we provide user guidelines, and finally, we present performance measurements of the *Sparqlify* tool against D2R [BC+06] and plain SQL queries on relational schema.

In Chapter 3, we present *TripleGeo*, a geospatial ETL tool for transforming several formats of geospatial data into RDF triples compliant with several standards. We enumerate the several input and output formats supported by the tool, we provide a user guide, a description of the system’s component and an experimental evaluation which focuses on comparing the efficiency of parsing and exporting RDF data in the the different supported output standards.

In Chapter 4, we conclude with some general remarks on the presented tools.

In the Appendix, we provide an example of triples extracted by utility *TripleGeo* from OSM datasets.
## Abbreviations and Acronyms

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BSBM</td>
<td>Berlin SPARQL Benchmark</td>
</tr>
<tr>
<td>CRS</td>
<td>Coordinate Reference System</td>
</tr>
<tr>
<td>DBMS</td>
<td>DataBase Management System</td>
</tr>
<tr>
<td>ETL</td>
<td>Extract-Transform-Load</td>
</tr>
<tr>
<td>GML</td>
<td>Geography Markup Language</td>
</tr>
<tr>
<td>GIS</td>
<td>Geographical Information Systems</td>
</tr>
<tr>
<td>KML</td>
<td>Keyhole Markup Language</td>
</tr>
<tr>
<td>LGD</td>
<td>LinkedGeoData</td>
</tr>
<tr>
<td>LOD</td>
<td>Linked Open Data</td>
</tr>
<tr>
<td>NLP</td>
<td>Natural Language Processing</td>
</tr>
<tr>
<td>OSM</td>
<td>OpenStreetMap (<a href="http://www.openstreetmap.org/">http://www.openstreetmap.org/</a>)</td>
</tr>
<tr>
<td>OWL</td>
<td>OWL 2 Web Ontology Language</td>
</tr>
<tr>
<td>RDBMS</td>
<td>Relational DataBase Management System</td>
</tr>
<tr>
<td>RDF</td>
<td>Resource Description Framework</td>
</tr>
<tr>
<td>RDFS</td>
<td>Resource Description Framework Schema</td>
</tr>
<tr>
<td>SPARQL</td>
<td>SPARQL Protocol and RDF Query Language</td>
</tr>
<tr>
<td>SQL</td>
<td>Structured Query Language</td>
</tr>
<tr>
<td>SRS</td>
<td>Spatial Reference System</td>
</tr>
<tr>
<td>SRID</td>
<td>Spatial Reference system IDentifier</td>
</tr>
<tr>
<td>SKOS</td>
<td>W3C Simple Knowledge Organization System</td>
</tr>
<tr>
<td>WKT</td>
<td>Well Known Text (as defined by ISO 19125)</td>
</tr>
<tr>
<td>WGS84</td>
<td>World Geodetic System (EPSG:4326)</td>
</tr>
<tr>
<td>W3C</td>
<td>World Wide Web Consortium (<a href="http://www.w3.org/">http://www.w3.org/</a>)</td>
</tr>
<tr>
<td>XML</td>
<td>Extensible Markup Language</td>
</tr>
</tbody>
</table>
# Table of Contents

1. **INTRODUCTION** .................................................................................................................. 8  
   1.1 KNOWLEDGE EXTRACTION ..................................................................................................................... 8  
   1.2 SPATIAL EXTRACT-TRANSFORM-LOAD (ETL) ........................................................................................ 9  
   1.3 CONVERTING GEOMETRIES INTO RDF .................................................................................................. 10  
   1.4 UPDATE WORKFLOW ANALYSIS ........................................................................................................... 11  

2. **SPARQLIFY** .......................................................................................................................... 13  
   2.1 TOOL DESCRIPTION ............................................................................................................................... 13  
      2.1.1 Implementation information ..................................................................................... 13  
      2.1.2 Components ................................................................................................................ 14  
      2.1.3 Integrated libraries ..................................................................................................... 14  
      2.1.4 Integrated data sources ............................................................................................. 15  
   2.2 MANUAL ................................................................................................................................................ 15  
      2.2.1 Building instructions ................................................................................................... 15  
         2.2.1.1 Sparqlify ....................................................................................................................... 15  
         2.2.1.2 Sparqlify-csv ................................................................................................................ 16  
         2.2.1.3 Sparqlify-platform ....................................................................................................... 16  
      2.2.2 API ................................................................................................................................ 16  
   2.3 THE LINDEDGEODATA USE CASE .......................................................................................................... 17  
      2.3.1 Translation tables ........................................................................................................ 18  
      2.3.2 Example scenario ........................................................................................................ 19  
      2.3.3 LGD database setup .................................................................................................... 19  
   2.4 PERFORMANCE EVALUATION .............................................................................................................. 20  
   2.5 FUTURE WORK ....................................................................................................................................... 23  
   2.6 LICENSING ............................................................................................................................................. 24  

3. **TRIPLEGEO** ........................................................................................................................ 25  
   3.1 TOOL DESCRIPTION ............................................................................................................................... 25  
      3.1.1 Implementation information .................................................................................................................. 25  
      3.1.2 Integrated tools/libraries ............................................................................................................ 26  
      3.1.3 Components .............................................................................................................................. 27  
   3.2 INTEGRATED DATA SOURCES ............................................................................................................... 28  
      3.2.1 Input data handling ....................................................................................................................... 28  
      3.2.2 Output data ............................................................................................................................... 28
3.3 MANUAL ............................................................................................................................................... 29
  3.3.1 Installation instructions .................................................................................................................. 29
  3.3.2 Configuration file .......................................................................................................................... 30
  3.3.3 Execution ....................................................................................................................................... 32
  3.3.4 Known limitations.......................................................................................................................... 32
    3.3.4.1 Handling large datasets ......................................................................................................... 32
    3.3.4.2 JDBC connection to geospatial DBMSs ................................................................................. 32
    3.3.4.3 Interacting with Oracle 11g databases on Linux platforms ..................................................... 33
  3.4 EVALUATION ........................................................................................................................................ 33
    3.4.1 Use case ....................................................................................................................................... 33
    3.4.2 Experimental validation ............................................................................................................ 34
  3.5 FUTURE WORK ..................................................................................................................................... 35
  3.6 LICENSING ......................................................................................................................................... 36

4. CONCLUSIONS ...................................................................................................................................... 37

5. REFERENCES .......................................................................................................................................... 38

6. APPENDIX ............................................................................................................................................... 43
List of Figures

Figure 1: The Sparqlify concepts and query rewriting workflow ................................................................. 13
Figure 2: Components of Sparqlify ............................................................................................................... 14
Figure 3: Overview of the new LinkedGeoData architecture ..................................................................... 17
Figure 4: SML view and a table as used in LinkedGeoData ......................................................................... 19
Figure 5: A SPARQL query that asks for tram stops in a specified area ..................................................... 19
Figure 6: Total benchmark runtime in seconds on a logarithmic scale ......................................................... 21
Figure 7: BSBM benchmark results comparing query runtime of BSBM queries ........................................ 22
Figure 8: SP2 benchmark query runtime on a 50k-triples dataset .............................................................. 23
Figure 9: Flow diagram of processing with utility TripleGeo .................................................................... 28

List of Tables

Table 1: Data contents of OSM layers (ESRI shapefiles) ........................................................................ 33
Table 2: Data contents of OSM layers (ESRI shapefiles) ........................................................................ 35
Table 3: Processing time when extracting triples from various geospatial repositories ........................... 35
1. Introduction

In this chapter, we briefly discuss related work on geospatial knowledge extraction and Extract-Transform-Load (ETL) processes. Specifically, we review the current standards and tools for extracting conventional, relational data into RDF formats. Then, we discuss well established ETL approaches for managing geospatial data. In the following, we briefly present two relevant initiatives in the (emerging) field of transforming geospatial data to RDF. Finally, we discuss possible workflows for dealing with updates on the initial dataset and their propagation to the produced RDF datasets. Overall, this chapter provides a background on the current state of art on geospatial and RDF data ETL and justifies the need of tools for transforming several input formats of conventional geospatial data into RDF triples. Two applications (Sparqlify and TripleGeo), with different and complementary functionality are developed in the frame of the current task and presented in the following chapters.

1.1 Knowledge Extraction

Creating knowledge from structured (e.g., relational databases, XML) or unstructured (e.g., text, documents, images) sources is a process known as Knowledge Extraction. Although it is methodically similar to Information Extraction (with Natural Language Processing tools, NLP) and Extract-Transform-Load (ETL tools for Data Warehouses), the main criteria is that the extraction result goes beyond the creation of structured information or the transformation into a relational schema, since it should facilitate inferencing. Hence, it requires either the reuse of existing formal knowledge (such as identifiers or ontologies) or the generation of a schema based on the source data.

Towards these goals, the RDB2RDF W3C group [RDB2RDF] has begun standardization of a language for extraction of RDF from relational databases. Another popular example for Knowledge Extraction is the transformation of Wikipedia [Wikipedia] into structured data and also the mapping to existing knowledge (e.g., DBpedia [DBpedia]). After the standardization of knowledge representation languages such as RDF and OWL, much research has been conducted towards transforming relational databases into RDF, also taking advantage of related research in domains such as Entity resolution, Knowledge Discovery and Ontology Learning. The general process uses traditional methods from Information Extraction and ETL, which transform the data from the sources into structured formats.

The R2RML W3C Recommendation [R2RML] specifies an RDF notation for mapping relational tables, views or queries into RDF. Such mappings provide the ability to view existing relational data in the RDF data model, expressed in a structure and target vocabulary of the mapping author’s choice. R2RML mappings are themselves RDF graphs and written down in Turtle syntax. R2RML enables different types of mapping implementations. Processors could, for example, offer a virtual SPARQL endpoint over the mapped relational data, or generate RDF dumps, or offer a Linked Data interface. The primary area of applicability is extracting RDF from relational databases, but in special cases R2RML could lend itself to on-the-fly translation of SPARQL into SQL, or to converting RDF data to a relational form, as this could serve for importing Linked Data into relational stores. This is possible if the constituent mappings and underlying SQL objects constitute updateable views in the SQL sense [LOD2KE].

About thirty (30) tools for Knowledge Extraction have been reviewed in [LOD2KE] in the realm of EU/FP7 project LOD2 [LOD2]. Although most of them are considered as proof-of-concept prototypes, more than ten of those tools are classified as “mature”, including Triplify [ADL+09], D2R Server [D2RServer], Ultrawrap [Ultrawrap], Virtuoso’s RDFizer Middleware (Sponger) [VirtSponger], and Virtuoso RDF Views
[VirtRDFViews], which enable transformation of relational databases into RDF. During the conversion process, these tools allow reuse of existing vocabularies and ontologies. In addition, the Google Refine RDF Extension [GRefineRDF] also seems especially promising, although it is under constant development. This extension can reconcile against SPARQL endpoints and RDF dumps, it offers a GUI for defining the shape of the RDF graph, and it can produce RDF under custom or imported vocabularies, hence it is expected to be highly useful for a multitude of use cases.

Overall, tools for converting data to and from RDF are becoming available. Fortunately, some of the more widely used semantic frameworks, such as Jena [Jena], can perform these file transformation operations. Still, there is a need to embed such RDF transformation utilities into mainstream ETL and data integration tools. One possible perspective is that this should be included by treating RDF as a native type when converting to/from files, databases and Big Data sources. However, to the best of our knowledge, none of the aforementioned methodologies and tools provides any particular support for geospatial data and operations.

1.2 Spatial Extract-Transform-Load (ETL)

Geospatial data may be located in a variety of different data formats, schemas, and heterogeneous platforms, systems, web services, etc. To meet such interoperability challenges, capabilities for spatial Extract, Transform, and Load must enable users to:

- Extract the spatial data from a source system.
- Transform the data into the format and data model required by the target system.
- Load the data into the target system.

Although ETL tools for processing non-spatial data have existed for some time, ETL tools that can manage the unique characteristics of spatial data did not emerge until the early 1990s. Such utilities are mainly being used for data cleaning, merging, verification or conversion into various formats.

Spatial ETL is sometimes referred to as data transformation or semantic data translation for geospatial information. The transformation phase of a spatial ETL process allows a variety of functions; some of these are similar to standard ETL, but some are unique to geospatial data. Indeed, this process may involve mapping of geometry and non-spatial attributes in the source data to geometry and attributes in the destination. It may include a change in coordinate system (i.e., reprojec tion), spatial feature types (e.g., modelling spatial interactions and calculating spatial predicates), topology (i.e., building topological relationships between disparate datasets), or modifying the attribute schema. Nonetheless, it always must preserve data integrity in order to provide consistent, accurate and well-defined information to end users.

In addition, a spatial ETL utility [SpatialETL] could potentially provide functionality for:

- Data Comparison, by carrying out change detection and doing incremental updates
- Conflict Management, for managing conflicts between multiple users against the same data
- Data Dissemination, in order to publish data on the Web or deliver by other means regardless of source format
- Semantic Processing, by understanding the rules of different data formats so as to minimize user input

The following software tools are among the most widely used for performing spatial ETL:

- **GOAL** is an open-source translator library for raster geospatial data formats, and implements the Simple Features data model for vectors in its OGR component [GDAL]. It can be applied for spatial ETL because it can convert between various proprietary storage models for geospatial
data (most spatial DBMSs, ESRI shapefiles, MapInfo TAB, AutoCAD DXF, GML, KML, etc.). However, no conversion to RDF triples is currently included in this package.

- **GeoKettle** is a powerful, metadata-driven Spatial ETL tool dedicated to the integration of different spatial data sources for building and updating geospatial data warehouses [GeoKettle]. Although GeoKettle supports a rich variety of spatial data formats and web services, it does not include any functionality to handle geospatial semantic data.

- Safe Software’s **FME Workbench** application is included with ESRI’s ArcGIS Data Interoperability extension [FMEworkbench]. It provides a visual diagramming environment that enables transformation of both geographic and attribute information. This Workbench application can be used either for creating custom formats to transform data on-the-fly or for developing custom spatial ETL tools for transforming data from new sources. These tools allow schema redefinitions and give the user full control of the translation and transformation process. However, FME Workbench does not include support for RDF data conversions.

1.3 Converting geometries into RDF

In terms of converting geospatial data into RDF resources, there seems to be very little interest in the research community. Only teams from Universität Leipzig and Universidad Politécnica de Madrid are active in this topic. More specifically:

i. **LinkedGeoData** [LGD] is an initiative from the research group Agile Knowledge Engineering and Semantic Web (AKSW) at Universität Leipzig towards adding a spatial dimension to the Web of Data / Semantic Web. This team has contributed a flexible system for mapping OpenStreetMap data to RDF, a SPARQL endpoint for making RDF data publicly available, as well as several tools for performing mappings and interlinking of geospatial semantic data. LinkedGeoData uses the information collected by the OpenStreetMap project [OSM] and makes it available as an RDF knowledge base according to the Linked Data principles. It interlinks this data with other knowledge bases in the Linked Open Data initiative. It currently consists of more than 1 billion nodes (i.e., geospatial points defined by a latitude and longitude) and 100 million ways (i.e., ordered lists of nodes that represent linear or polygon features). The resulting RDF data comprises approximately 20 billion triples. The data is available according to the Linked Data principles and interlinked with DBpedia [DBpedia] and GeoNames [GeoNames].

ii. The Ontology Engineering Group (DIA) of the Facultad de Informática at Universidad Politécnica de Madrid has also worked on **Geo.LinkedData.es** [GLD], which is an open initiative to enrich the Web of Data with Spanish geospatial data. This initiative started off by publishing diverse information sources belonging to the National Geographic Institute of Spain. Such sources are made available as RDF knowledge bases according to the Linked Data principles. Such data is interlinked with other knowledge bases belonging to the Linking Open Data Initiative. This team has made available two tools that enable conversion of geospatial data into RDF. The first, **map4rdf** [map4rdf] is an open source software that can be configured to use a SPARQL endpoint and provide map-based visualization of data. The geospatial aspects of the data can be modelled using either the data model from W3C Geo XG or the geometrical data model proposed by this research group (prior to, and thus not compliant with the GeoSPARQL standard [OGC12]). Among its features, there are included a faceted browser interface, visualization on top of Google Maps or Open Street Maps, editing and storing displayed data in RDF format, query filtering, etc. The second tool, **geometry2rdf** [geo2rdf] enables extraction of geometries into RDF triples. As this library has provided the source base for developing our own tool **TripleGeo**, it is reviewed in more detail in Section 3.
1.4 Update workflow analysis

In general, an update workflow is a sequence of steps that need to be performed in order to propagate changes from (a set of) source datasets to (a set of) target datasets. A comprehensive survey of approaches for dataset dynamics in the Semantic Web is described in [UVH+10], which serves as a base for this section. In the following, we summarize important aspects of dataset dynamics as well as corresponding approaches, protocols and technologies.

**Resource dynamic description.** A dataset may carry a description about its dynamics, such as change frequency (e.g. hourly, daily, monthly), last modification date, and version. The Sitemap protocol [SMAPS], originally developed by Google, enables one to describe resources that exist on a web site as well as their dynamic properties. In practice this is used for effective discovery and indexing by web crawlers, such as those used by search engines. See [UVH+10] for a list of several vocabularies which offer a Semantic Web way of expressing similar information. Note that protocols usually cover multiple aspects of dataset dynamics.

**Change detection/notification.** There may be services that enable consumers to detect whether a change to a dataset occurred at all. This can happen either via pull (polling) or push (triggering) approaches. Depending on the used protocol, a resource dynamic description may already serve as a means for change detection, e.g. by providing a last modification date. In general, one can distinguish between pull and push approaches:

- **push:** These are publish/subscribe approaches, where consumers register themselves as listeners to a notification service. Whenever a change occurs, the listeners are notified. Protocols considered for this purpose are PubSubHubbub [FSAG+13], Websocket [WSP], Pingback [PING] and its semantic extension Semantic Pingback [TFEA+10]. Database engines may also come with their own replication system, such as Virtuoso’s subscription/notification system [VIRTrepl].
- **pull:** These approaches require a consumer to continually check for occurred changes. Technologies include Atom [ASF], the Open Archive Initiative - Protocol for Metadata Harvesting, (OAIPMH) [OAIP], and Sitemaps.

**Change description.** A change description may capture metadata about the change itself (e.g. why, when and by whom the change was made) and effect of a change (affected classes, but possibly bounding boxes or (administrative) boundaries). The change description can be either delivered with the change, or provided separately.

**Change retrieval and transport.** Consumers need to be able to retrieve the actual information about what has changed. Depending on which approach was used, the actual change information may be transferred as a payload in one of the aforementioned communication protocol messages.

**Change information content and representation.** There are several ways of how changes can be represented:

- **RDF based** Provisioning of added and removed triples in one of the (quasi) standard RDF syntaxes. A list of ontologies together with their supported granularities (resource, triple or graph) can be found in [UVH+10].
- **Diff based.** The diff syntax (the output of the diff program) can mark blocks of lines which were inserted or removed. In the case of N-Triple syntax, each block directly corresponds to a set of added or removed set of triples.
- **Ontology based.** Changes can be represented as entities modeled in RDF. An "Update Ontology", provides a vocabulary for modeling certain types of changes.

- **SPARQL 1.1 Update based.** With SPARQL 1.1, a DML for RDF data became standardized. Using INSERT DATA/DELETE DATA, any extensional update operation can be readily represented as a SPARQL query. Intensional updates can be represented by INSERT/DELETE and MODIFY statements.

- **Programmatic based.** SPARQL 1.1 features a great expressivity for many use cases, however, it is not turing complete. Therefore, some update processes may need to be done programmatically. Although eventually, such a program eventually has to compute sets of added and removed sets of triples, it is the application logic that decides on which changes to perform.

**Change analysis.** Based on business requirements, it may be necessary to postpone or reject certain changes. For example, the application of updates from OpenStreetMap that break previously valid administrative boundary polygons may be undesired. Potential problems need to be reported.

**Change verification.** Based on the analysis, decide on whether and which changes can be applied.

**Change application.** The change is applied to the dataset. This could be an insert in a database or an update of a file. All RDF stores are capable of applying changes, whereas SPARQL 1.1 Update is a standard language for this purpose. As RDF model based diff application similar to rsync is RDFSsync [TMBE+08], which offers specific optimizations by taking RDF semantics into account.

**Change publishing and propagation.** A target dataset that has been updated from one or more source datasets may in turn act as a source dataset for further processing steps. Hence, it may be neccessary to notify any clients listening for changes, for reasons such as cache invalidation and the contition of an update chain/workflow. This are the items Change detection/notification and Change retrieval and transport from a data set publisher perspective. Therfore, aforementioned protocols are applicable.

DBpedia Live [MLASH+12, DBPlive] is a well known example of a LOD data set which processes changes from a Wikipedia and publishes corresponding change sets in RDF: In a nutshell, the system fetches sets of modified Wikipedia articles via the OAI-PMH interface, extracts the articles' corresponding triples, compares them to the prior extracted triples already in the production database, and writes out changesets in form of two files with added and deleted triples at regular intervals. These changesets are then applied to the DBpedia Live database and published as downloads for consumers.
2. Sparqlify

In this section we present **Sparqlify**, a SPARQL-to-SQL rewriter which enables SPARQL queries on relational databases.

2.1 Tool description

**Sparqlify** is a novel SPARQL-SQL query rewriter that greatly simplifies the definition of RDF views thanks to the intuitive **Sparqlification Mapping Language** (SML). Whereas most current mapping approaches use RDF and XML as a means to represent the mapping information, Sparqlify mappings are expressed as view definitions based on the **SPARQL** grammar [SPARQLgram] that has been extended with a few custom production rules. As such, users that know SPARQL are already familiar with most of Sparqlify’s syntactic elements. The remainder of this section gives an overview of Sparqlify’s implementation, its usage, and showcases its deployment in the LinkedGeoData project [LGD].

2.1.1 Implementation information

In general, SPARQL-to-SQL rewriters are configured with a database and a set of RDB-to-RDF view definitions. Based on this configuration, Sparqlify compiles subsequent SPARQL queries into two related artifacts: An **SQL query** and a **binding** of SPARQL variables to expressions over the SQL query’s result set. Figure 1 shows the steps of Sparqlify’s query rewriting process, which are summarized as follows.

![Figure 1: The Sparqlify concepts and query rewriting workflow.](image-url)
First, the query is converted into an algebra expression. This expression is subsequently converted to a normal form. Given the query patterns, relevant Sparqlify-ML views need to be detected. After this is done, the algebra expression is rewritten to include those relevant views. In a next step, optimisations on the algebra expression are performed to improve efficiency. Finally, this algebra expression can be transformed to an SQL algebra expression. For accomplishing this, we define a general relational algebra for RDB-to-RDF mappings. The SQL query, which was obtained, is executed against the relational database. Using the defined mappings, the SQL result set returned by the relational database can be converted to a SPARQL result set.

All of the above steps are explained in detail in the technical report [SULA+13].

2.1.2 Components

Sparqlify is comprised of the following components, whose relations are depicted in Figure 2.

- At the core there is the Sparqlify engine, written in Java, which carries out the actual SPARQL-to-SQL rewriting.
- The Sparqlify server provides the HTTP web interface.
- The Sparqlify platform is an integration project that currently bundles the Sparqlify server with the Linked Data interface Pubby [PUBspar] (the original Pubby project is hosted under [PUB]) and the SPARQL web front end Snorql [SNORQL].

![Figure 2: Components of Sparqlify](image)

2.1.3 Integrated libraries

Sparqlify depends on various open source libraries. An excerpt is given in this section. Note that Sparqlify is a Maven [MAVEN] project, and thus its complete set of dependencies is declared in the project’s `pom.xml` [SPRQLFpom].

- **Apache Jena.** A popular, powerful and mature Java Semantic Web toolkit [JENAapa].
- **Jersey.** A web framework for the creation web applications (both RESTful and stateful) [JERS].
- **Atmosphere** [ATM]. An extension of Jersey offering support for various server side push techniques. Although Sparqlify does not make use of these additional features yet, having this support already in place may ease the addition of future features, such as an integration of our SparqlAnalytics project [SPARanal].
- **Google Guava.** A general purpose Java library. In our context mainly used for its collections [GUAVA].
- **Simple Logging Facada for Java (SLF4J).** A logging framework [SLF].
2.1.4 Integrated data sources

The Sparqlify project integrates the R2RML test suite [R2RML].

2.2 Manual

This section explains the setup and deployment procedure for Sparqlify. As Sparqlify is under active development, we recommend to check the project's Github page [SPRQLF] for the latest state. A bleeding edge Debian package is automatically generated and pushed to a repository [SPRQLFdeb] on every successful build after a commit. Stable releases and release candidates are manually published [SPRQLFrel].

2.2.1 Building instructions

Because Sparqlify is Open Source, it can also be built from source. For this, the following commands need to be run:

```bash
git clone https://github.com/AKSW/Sparqlify
cd Sparqlify
mvn clean install
cd sparqlify-core
mvn assembly:assembly
```

Afterwards, the scripts should be working.

```bash
sparqlify -core/bin/sparqlify
sparqlify -core/bin/sparqlify -csv
sparqlify -platform/bin/sparqlify -platform
```

Alternatively, the debian package can be built using:

```bash
# At <repository -root >
mvn clean install
cd sparqlify -debian
mvn clean install deb:package
```

This creates a .deb file under `sparqlify-debian/target/`.

2.2.1.1 Sparqlify

Usage: sparqlify [options]
Options are:

- **Setup**
  - `s` SML view definition file

- **Database Connectivity Settings**
  - `-h` Hostname of the database (e.g. `localhost` or `localhost:5432`)
  - `-d` Database name
  - `-u` User name
  - `-p` Password
  - `-j` JDBC URI (mutually exclusive with both `-h` and `-d`)

- **Quality of Service**
  - `-n` Maximum result set size
  - `-t` Maximum query execution time in seconds (excluding rewriting time)

- **Stand-alone Server Configuration**
  - `-P` Server port [default: `7531`]

- **Run**
  - `-o` Once (these options prevent the server from being started and are mutually exclusive with the server configuration)
  - `-D` Create an N-TRIPLES RDF dump on STDOUT
2.2.1.2 Sparqlify-csv

Usage: sparqlify -csv [options]

Setup
- `-m` SML view definition file
- `-f` Input data file
- `-v` View name (can be omitted if the view definition file only contains a single view)

CSV Parser Settings
- `-d` CSV field delimiter (default is `'''`
- `-e` CSV field escape delimiter (escapes the field delimiter) (default is `'\''`
- `-s` CSV field separator (default is ',')
- `-h` Use first row as headers. This option allows one to reference columns by name additionally to its index.

2.2.1.3 Sparqlify-platform

The Sparqlify Platform (under /sparqlify-platform) bundles Sparqlify with the Linked Data wrapper Pubby and the SPARQL Web interface Snorql.

Usage: sparqlify -platform config -dir [port]
  config -dir Path to the configuration directory, e.g. `<repository -root /sparqlify-platform/config/example >`
  port Port on which to run the platform, default 7531.

For building, at the root of the project (outside of the sparqlify-* directories), run `mvn` compile to build all modules. Afterwards, launch the platform using:

```
cd sparqlify-platform/bin ./sparqlify-platform <path-to-config> <port>
```

Assuming the platform runs under `http://localhost:7531`, the following services are available relative to this base url:

```
/sparql is Sparqlify 's SPARQL endpoint
/snorql shows the SNORQL web frontend
/pubby is the entry point to the Linked Data interface
```

2.2.2 API

The standalone `sparqlify` script starts a SPARQL endpoint at the configured port (7531 by default) and can be accessed under at:

```
http://localhost:7531/ sparql
```

Currently the query string parameters `query` and `default-graph-uri` are supported. The `sparqlify-platform` runs under the context path `sparqlify-platform`, where it provides the following resources:

```
http://localhost:7531/ sparqlify-platform/sparql
http://localhost:7531/ sparqlify-platform/snorql
http://localhost:7531/ sparqlify-platform/pubby/<path >
```

The SPARQL web interface runs under `snorql`. All resources created from the view definitions that correspond to HTTP URIs are automatically made available under `/pubby`. This means that resources, such
as http://resource-hostname/path, are exposed as http://sparqlify-platform-hostname/pubby/path.

For publishing these Linked Data resources on the Web, their URLs can be further rewritten using reverse proxy techniques as offered by most web servers. For an example, see the documentation of the deployment of Sparqlify at Panlex [PNLX].

2.3 The LinkedGeoData use case

The LinkedGeoData (LGD) is an effort to add a geospatial dimension to the Semantic Web. LGD uses the information collected by the OpenStreetMap community and makes it available as an RDF knowledge base according to the Linked Data principles. In this section, we explain our approach to mapping the OpenStreetMap (OSM) database to RDF.

Instances of the OSM database can be created with the Osmosis tool [OSMOSIS]. Osmosis ships with two schemas for the PostgreSQL database, namely simple and snapshot [OSMschema]. Both schemas are capable of storing the same information. However, snapshot uses the special hstore datatype [HSTORE], which enables the storage of sets of tags in a single row. Since this datatype defines a special set of operators, it is not yet supported by Sparqlify (and to the best of our knowledge neither by any other RDB-RDF mapping tool). For this reason we used the purely relational simple schema. Figure 3 depicts the architecture of LGD.

![Diagram](image-url)

**Figure 3: Overview of the new LinkedGeoData architecture.**
At the core, the LinkedGeoData database (PostgreSQL/PostGIS) is comprised of:

- All tables from the *simple* schema. The essential ones being **nodes**, **ways** and **relations**. These tables have a primary key of type bigint, and additional columns for the metadata about the respective OSM entities, such as the user ID and the date of the most recent change. For each table there exists also a corresponding tag table with the structure \{\text{way, node, relation}_id, k, v\}.
- Several **translation tables** that are used to store the mappings between OSM tags and RDF resources. In order to avoid confusion with the RDB-RDF mappings, we use the term "(tag) translation" to refer to the concept of translating OSM tags to RDF terms.
- A set of SQL helper views that hide underlying complexity. The complexity stems from joining the simple schema with the translation tables, and the use of function indexes on tags whose values can be converted to numeric values.
- Additional indexes for the keys and values of the OSM tags. Furthermore we use functional indexes to capture those tag-values that can be parsed as numerics.

Additionally, there are tables for the following data that was integrated from external sources:

- OWL **SameAs links** to DBpedia and GeoNames.
- Icons from SJJB Management [SJJBicons].
- Multilingual labels for the schema of the LinkedGeoData ontology obtained from the OSM translation projects coordinated on TranslateWiki [TRWI].

### 2.3.1 Translation tables

Whereas the subject of the triples generated from the tag tables is determined by an OSM entity's ID, their predicates and objects depend on the tags. The following tables are used in LGD to store the information that map tags \( (k, v) \) to RDF.

- **lgd_map_datatype**: Associates a key with a numeric datatype (integer or real).
- **lgd_map_label**: Associates \((k, v)\) with \((\text{label, languageTag})\).
- **lgd_map_literal**: Maps \( k \) to a property, and transforms \( v \) into a literal with specified language tag.
- **lgd_map_resource_k**: Any entity having a key \( k \) is mapped to a pair of URIs for the property and object.
- **lgd_map_resource_kv**: Same as above, except that both \( k \) and \( v \) must exist.
- **lgd_map_property**: Covers cases, where \( v \) is already a URI, and only \( k \) needs to be mapped to a property. Used for keys such as *website* or *depiction*.

Figure 4 shows how a translation table is mapped to RDF. The view maps tags to classes, and additionally exposes for each class the information from which tag it was created. For each row of the underlying table an instance of an OWL class is created. This instance is then annotated with the original tag it was created from. The complete view definition file is available on Github [SPRQLFview].
With

```
?s = uri(?object)
?t = uri(concat(?object, "/key/",
    ?k, "/value/", ?v))
?k = plainLiteral(?k)
?v = plainLiteral(?v)
```

Constrain

```
?s prefix "http://linkedgeodata.org/ontology/"
?t prefix "http://linkedgeodata.org/ontology/"
```

From

```
[[SELECT object, k, v FROM
  lgd_map_resource_kv WHERE
  property = 'rdf:type']]
```

<table>
<thead>
<tr>
<th>lgd_map_resource_kv</th>
</tr>
</thead>
<tbody>
<tr>
<td>k</td>
</tr>
<tr>
<td>amenity</td>
</tr>
<tr>
<td>route</td>
</tr>
<tr>
<td>sport</td>
</tr>
</tbody>
</table>

Figure 4: SML view and a table as used in LinkedGeoData.

2.3.2 Example scenario

Figure 5 shows a use case of LGD for finding tram stops within a certain region (it can be run on [SNORQLexpl]). The spatial predicate `ogc:intersects` and the function `ogc:geomFromText` translate eventually to the PostGIS's functions `ST_Intersects` and `ST_GeomFromText`. The underlying view definition's `WITH` clause only contains an entry

```
?g = typedLiteral(?geom, ogc : WKTLiteral)
```

for mapping the geometry type. The type is then handled appropriately by the Sparqlify engine. Due to the the mapping process with Sparqlify, LGD also exposes information about OSM relations, such as the administrative boundaries [EXBG], or which tram stops belong to which tram routes. The full compatibility of the LGD schema with the OSM simple schema also shows, that Sparqlify fits nicely into the OSM tool chain.

```
SELECT * WHERE {
  ?b a lgdo:TramStop .
  ?b rdfs:label ?l .
  ?geom ogc:asWKT ?geo .
  Filter(ogc:intersects(?geo,
    ogc:geomFromText('POLYGON((8 50, 12 50, 12 55, 8 55, 8 50)))'))
} Limit 100
```

Figure 5: A SPARQL query that asks for tram stops in a specified area.

2.3.3 LGD database setup

The easiest way to set up an LGD database is to install the Sparqlify and LinkedGeoData debian packages, [SPRQLFdeb] and [LGDdeb] respectively. The corresponding sources are available at [LGDgit and [SPRQLFgit].

---

1 For convenience we tied this function to the ogc namespace.
Afterwards, a couple of tools for creating and querying an LGD database are available. These tools assume the following defaults in regard to database connectivity:

- **host name**: localhost
- **user name**: postgres
- **password**: postgres
- **database name**: lgd

The following command sets up a PostGIS database with the OSM schema and the LGD extensions:

```
lgd-createdb
   -h postgres host name
   -d postgres database name
   -U postgres user name
   -W postgres password (will be added to ~/.pgpass if not exists)
   -f .pbf file to load (other formats currently not supported)
```

Example:

```
wget http://download.geofabrik.de/europe/germany/bremen-latest.osm.pbf
lgd-createdb -h localhost -d lgd -U postgres -W mypwd -f bremen-latest.osm.pbf
```

The following command runs a SPARQL query on the database. Internally, Sparqlify is called.

```
lgd-query
   -h postgres host name
   -d postgres database name
   -U postgres user name
   -W postgres password (will be added to ~/.pgpass if not exists)
   -Q SPARQL query string or named query
```

A named query is just a SPARQL query that is referenced by a name. The mapping of a name to a SPARQL is configured via `lgd.conf.dist`. Currently, the following named queries exist:

- **ontology**: Creates an N-Triple output with all classes and properties
- **dump**: Create a full dump of the database

Examples:

```
lgd-query ontology
lgd-query dump
```

### 2.4 Performance evaluation

In this section, we discuss the evaluation of Sparqlify using the Berlin Sparql Benchmark (BSBM) [BS+09] and SP2Bench (sp2b) [SHLP+08]. As a baseline tool well known tool D2R [BC+06] was chosen because of its popularity. We selected BSBM because of its widespread use and the provision of both RDF and relational data with corresponding queries in SPARQL and SQL. We used the Explore test scenario,
consisting of 11 SPARQL queries and their SQL translations. This Explore scenario simulates a web shop in which a user browses the product catalog to search for specific products. Datasets were generated with sizes of 1, 25 and 100 million triples. The queries were performed on a single thread with 10 warm-up and 100 measurement runs, using the mapping provided on the BSBM website [BSBMmap]. In addition to the original BSBM queries, we created a refactored query set. In these queries the resource URI by which BSBM parameterizes a query were factored out of triple patterns and expressed in an equivalent FILTER condition. The factorized queries therefore have the same result sets as the original queries, but reflect a different usage of query patterns.

The sp2b benchmark simulates the analysis of a bibliographic database. In contrast to BSBM, the queries of sp2b are more analytical, with longer runtimes, complex query structures and unbound result set sizes. Due to their complexity, of the 17 queries provided only 9 queries could be executed on both D2R (9 working queries) and Sparqlify (10 working queries). The dataset used in the sp2b run is sized 50,000 and 250,000 triples. The sp2b run was performed with 5 warmup runs and 10 measurement runs.

We used a virtual machine with three AMD Opteron 4184 cores and 4 GB RAM allocated for measurements. Data resided in all cases in a PostgreSQL 9.1.3 database [PostgreSQL], with 1 GB of RAM. Additional indices to the BSBM relational schema were added, as described in the BSBM results document [BSBMres]. Likewise, we added indices into the sp2b database for all columns generating URIs. Both D2R and Sparqlify were allocated 2 GB of RAM. We utilized D2R version 0.8 and activated in all cases the optimized fast mode.

Figure 6: Total benchmark runtime in seconds on a logarithmic scale

Figure 6 shows the time totals required for completing the benchmark runs on the different datasets, comparing Sparqlify, raw SQL and D2R with fast mode. A first observation is the clear dominance of SQL over D2R and Sparqlify in the smaller datasets. We attribute the slower execution time of SPARQL queries to the overhead imposed by the query translation. With increasing dataset size in the case of Sparqlify this overhead is less significant and consequently the performance gap closes. When relating the execution time of the SPARQL-mapper with SQL, the superior scaling behavior of Sparqlify can be observed. The factor by which Sparqlify is slower than SQL decreases from 28 (1m) to 3.9 (25m) and 3.3 (100m). For D2R, this factor increases from 111 (1m) to 304 (25m). The 100 million triple dataset could not be benchmarked, as D2R was running out-of-memory [SourceD2R]. An additional interesting observation is the increase in execution time for the factorized queries. Due to the normalization step Sparqlify is essentially not affected by such purely syntactical changes of a query, whereas this is the case for D2R.

Using sp2b similar results can be observed. Sparqlify is faster by a factor of 4.5 on the 50k dataset in this case. For the 250k dataset, D2R and Sparqlify both failed on different queries as shown in Figure 8.
Examining the benchmark results on a per-query basis, as presented in Figure 7 for BSBM, allows a more detailed discussion of the results. The query execution times of Query 7 and 8 are the main reasons why D2R falls behind Sparqlify in both the 1m and 25m triples scenario. These two queries cause D2R to fetch huge intermediate result sets from the database, which are then processed internally. In the 100m benchmark scenario these result sets do not fit into memory any more and thus cause the out-of-memory exceptions. Sparqlify, on the other hand, does not suffer from this problem, as it only generates a single SQL query and therefore puts the whole workload on the SQL database. As such, it achieves a performance comparable to the SQL counterpart.

![Figure 7: BSBM benchmark results comparing query runtime of BSBM queries.](image)

An interesting observation is related to Query 5, which in its SQL variant is one of the most expensive queries. The corresponding query execution times for Sparqlify and SQL converge with increasing dataset size, which shows that the overhead of the mapping process becomes less significant.

Also noteworthy is, that while most of the queries are faster on D2R, the difference in performance can be attributed to the larger overhead imposed by the Sparqlify approach. This penalty is most visible in the smaller, fast queries of BSBM. A conclusion is that the overhead imposed by the previously described...
formal framework can have negative effects for smaller datasets and simpler queries, but offers advantages with growing dataset size and query complexity. The results of sp2b, as displayed in Figure 8 fit well into this picture.

The generally higher query executions times of the sp2b queries (minimum 0.05s) reduce the effect of the mapping overhead. In the sp2 benchmark, Sparqlify therefore shows superior performance compared to D2R.

For the 50k sp2b experiment, Sparqlify prevails for all queries with the exception of 3b and 12a. For the other queries, expensive round-trips between the mapper and the database lead to performance penalties for D2R. The second benchmark run on a 250k dataset demonstrates the limits of D2R with complex queries. Here 6 of 9 queries fail because of running out of memory, whereas Sparqlify manages to perform 7 out of 9 queries. Sparqlify fails in 2 queries of sp2b because for them PostgreSQL's query execution plan involves storing huge amounts of data in temporary tables.

In summary, D2R outperforms Sparqlify on smaller datasets and less complex queries. However, Sparqlify provides better scalability on large datasets, which is crucial for the mapping large crowdsourced datasets as in the LinkedGeoData use case.

![Figure 8: SP2 benchmark query runtime on a 50k-triples dataset.](image)

2.5 Future work

There are several directions for future work for both Sparqlify and LinkedGeoData. Based on our experience with R2RML and SML, we claim that the latter is easier to learn and use, and thus offers better productivity. Therefore we will continue work on an SML specification and also develop an evaluation tool in order to collect empirical evidence. Our vision for Sparqlify is, for it to support the full SQL MM standard; therefore more work will be done on Sparqlify’s geospatial features. Another topic that needs to be investigated is how Sparqlify can be moved towards conformance with the GeoSPARQL standard, which implies the requirement of having limited reasoning support. In regard to LinkedGeoData, besides adding more links to other datasets, a future step is to investigate how polygons can be efficiently aggregated and published in realtime from the incoming changesets.
2.6 Licensing

The following licenses apply:

- Sparqlify: Apache License Version 2.0 [SPRQLFlic].
- LinkedGeoData: The LinkedGeoData conversion project is licensed under GPL v3 License [LGDlic]. The datasets are published under the Open Database License (ODbl) [DATAlic].
3. TripleGeo

In order to use geospatial data in triple stores, the respective geometries of the original datasets have to be converted into a format recognisable by each platform. Despite the recently published GeoSPARQL standard [OGC12], only a few triple stores have currently implemented its specifications (e.g., Parliament, uSeekM). Instead, several triple stores with geospatial support prefer proprietary geometric representations (e.g., AllegroGraph) or restrict their focus to points only (e.g., Virtuoso, OWLIM-SE), as already analyzed in Deliverable D2.1.1 of this project [GeoKnowD21].

In order to provide a tool for integrating geospatial features from databases, we have developed a generic purpose utility, called TripleGeo [TripleGeo], which is based on the open-source utility geometry2rdf [geo2rdf]. This earlier tool has been substantially modified and enhanced to extract non-geographical attributes and also interact with diverse geographical and triple formats.

In fact, TripleGeo can take as input not only ESRI shapefiles, but may also access spatial tables hosted in major DBMSs (e.g., Oracle Spatial or PostGIS databases). Further, it copes with most common spatial data types, like points, linestrings and multi-linestrings, as well as polygons and multi-polygons. In addition, TripleGeo can make on-the-fly transformation of a given dataset into another projection system (e.g., data from a national reference system like GreekGrid87 into WGS84). Geometries can be exported in several serialized formats, most typically in WKT as prescribed by the recent GeoSPARQL standard [OGC12]. However, there is also support for namespace pos: of the WGS84 GeoPositioning vocabulary [GeoPos84] and Virtuoso’s namespace for custom point geometries virtrdf:.

Execution of command-line utility TripleGeo is parameterized with a configuration file that declares user preferences concerning the conversion. When initiated, this process iterates through all features in the original dataset and emits a series of triples per record. Every geometric feature is turned into properly formatted triple(s), according to the specified vocabulary. Additional descriptive attributes can be extracted as well, including identifiers, names, or types. For the time being, such attributes are exported as literals, without taking into account any underlying ontology. Results are written into a file in a standard format for triples, like RDF/XML, Turtle, N-triples, etc., so that they can be readily imported into a triple store.

In this Section, we first describe how TripleGeo has been implemented, by examining its components and processing flow, along with its dependencies on third-party platforms and libraries. Then, we provide information about supported input and output representations, with particular emphasis on geometries. We offer a short manual with important execution details, as well as some known limitations of the software. Next, we report our practical experience after testing this utility against several types of geometries stored in various geospatial repositories. We conclude with directions for further extensions planned for future releases of the software, along with licensing information.

3.1 Tool description

3.1.1 Implementation information

geometry2rdf [geo2rdf] is an open-source library developed in Java by the Ontology Engineering Group (DiA) of the Facultad de Informática at Universidad Politécnica de Madrid. This tool allows the definition of geometrical information in RDF format. This methodology, also proposed in [VVS+10] for handling linked geodatasets, relies on Oracle’s SDO_UTIL package for transforming geometrical data into
GML format. For geometries stored in a MySQL database, information from the GEOMETRY column is extracted in a WKT representation. The next step is to convert the generated GML into RDF. For this purpose, the team has developed the `geometry2rdf` library, which exports a set of RDF triples with geometrical information. Geometries can be available in GML or WKT and are manipulated with GeoTools [GeoTools], not only in order to retrieve features, but also to perform coordinate transformation (if required). Finally, the Jena Semantic Web Framework [Jena] is used to generate the final geospatial RDF. The RDF generated is compliant with the WSG84 RDF vocabulary [GeoPos84] and the GML ontology [OGC07].

It is important to note that currently (in the latest version of `geometry2rdf` available at [geo2rdf]) there is no support for geometries in the GeoSPARQL standard [OGC12], and no capability to export in formats other than RDF. In addition, there is no support for handling attribute values related to features (e.g., names, types). Concerning interaction with geospatial DBMS platforms, only support for extracting geometries from ESRI shapefiles and Oracle Spatial is available. Despite these important deficiencies, this source code provided a stable base for developing our own utility, mostly geared towards integration of several database platforms and support for GeoSPARQL types as explained in the following.

### 3.1.2 Integrated tools/libraries

**TripleGeo** inherits from `geometry2rdf` dependencies to various open-source tools and libraries, all of which are used “as is”. The most significant of these libraries are:

- **Apache Jena**. This is a Java framework for building Semantic Web applications. Jena [Jena] provides a collection of tools and Java libraries for developing semantic web and linked-data apps, tools and servers. In April 2012, Jena graduated from the Apache incubator process and was approved as a top-level Apache project. The Jena Framework includes:
  - an API for reading, processing and writing RDF data in XML, N-triples and Turtle formats;
  - an ontology API for handling OWL and RDFS ontologies;
  - a rule-based inference engine for reasoning with RDF and OWL data sources;
  - a query engine compliant with the latest SPARQL specification; and
  - servers that allow RDF data to be published to other applications using a variety of protocols, including SPARQL.

- **GeoTools**. GeoTools [GeoTools] is an open source (LGPL) Java library, which provides standards compliant methods for geospatial data management comparable to those implemented in Geographical Information Systems (GIS). The GeoTools library implements Open Geospatial Consortium (OGC) specifications such as ISO 19107 Geometry, Simple Features, Clients for Web Feature Service (WFS) and Web Map Service (WMS), etc. GeoTools is widely used by a number of projects including Web Feature Servers, Web Map Servers, and GIS desktop applications. Among its core features are included:
  - Definition of interfaces for key spatial concepts and data structures, such as Integrated Geometry support provided by Java Topology Suite (JTS), attribute and spatial filters using OGC Filter Encoding specification, etc.
  - A clean data access API supporting feature access in many file formats (like CSV, DXF, edigeo, excel, GeoJSON, Shapefile, WFS) and spatial databases (including DB2, H2, MySQL, Oracle Spatial, PostGIS, Spatialite, MS-SQL Server), as well as coordinate reference system and transformation support, an extensive range of map projections, transaction support and locking between threads, etc.
  - A low-memory renderer, to compose and display maps with complex styling.
A schema-assisted parsing technology using XML Schema with bindings for many OGC standards including GML, KML, etc.

- Plug-ins for reading additional raster formats from GDAL.
- Extensions providing graph and networking support, validation, a web map server client, bindings for XML parsing and encoding, etc.

**GDAL/OGR** The Geospatial Data Abstraction Library [GDAL] is a translator library for raster geospatial data formats, supported by the Open Source Geospatial Foundation (OSGeo). As a library, it presents a single abstract data model to the calling application for all supported formats. It also comes with a variety of useful command-line utilities for data translation and processing. The related OGR Simple Features Library is a C++ open source library (which lives within the GDAL source tree) and provides similar capabilities (and command-line tools) for read (and sometimes write) access to a variety of vector file formats including ESRI Shapefiles, PostGIS, Oracle Spatial, Mapinfo mid/mif and TAB formats, etc.

**Java Topology Suite (JTS)** The JTS Topology Suite [JTS] is an API of 2D spatial predicates and functions, conforming to the OGC Simple Features Specification for SQL. JTS is open source (under the LGPL license) and provides a complete implementation of fundamental 2D spatial algorithms written in Java.

### 3.1.3 Components

TripleGeo has been implemented with several Java classes that perform specific tasks in a modular fashion. From a user’s perspective, the utility works in an opaque fashion according to some preconfigured settings. Figure 9 presents the flow diagram used for converting geospatial features into triples. Next, we outline the basic components of the utility:

- **Input geospatial data** may be obtained either from ESRI shapefiles or directly from geospatially-enabled DBMSs. Currently, TripleGeo can access features stored in Oracle Spatial, PostGIS, MySQL, and IBM DB2 databases.
- **Connectors** to source data are required in order to access geometric features. In case of a DBMS, this is possible thanks to suitable JDBC drivers. With respect to shapefiles, the integrated GeoTools library provides all required functionality.
- A **configuration file** lists properties that control several stages: how input source will be accessed, which data is involved, what geometric representation should be used, whether geometries must be transformed in another reference system, as well as the output format. All properties that may be specified in this file are explained in Section 3.3.2.
- A **parser** iterates through each input record and converts geometries into a suitable representation according to user specifications. It also consumes non-spatial attribute values (e.g., types, names) of the features involved and emits properly formatted literals.
- A **jena model** is a main-memory data structure that is used to retain all state information consisting of the collection of generated triples. This model denotes an RDF graph, so called because it contains a collection of RDF nodes, attached to each other by labelled relations. In Java terms, this model as the primary container of RDF information contained in graph form. A significant benefit from using the Jena model is that it offers a rich API with many methods intended to make it easier to write RDF-based applications.
- Optionally, **reprojection** of geometries into another spatial reference system is available. This transformation is carried out thanks to the integrated GeoTools library and according to user specifications for the source and target CRS.
- **Export** of generated triples into files is performed by the Jena API. This offers the possibility of writing the output into several triple formats, as detailed in Section 3.2.2.

The resulting triples are written into a single file at the path specified in the configuration settings.
### 3.2 Integrated data sources

#### 3.2.1 Input data handling

The current version of **TripleGeo** utility can access geometries from:

- ESRI shapefiles (ESRShp), which is a well-known format for storing geospatial features in files.
- Several geospatially-enabled DBMSs, including:
  - Oracle Spatial [Oracle];
  - PostGIS (spatial module for PostgreSQL) [PostGIS];
  - MySQL [mySQL];
  - IBM DB2 with Spatial extender [IBM-DB2].

Each data source may contain several attributes (i.e., table columns), but at most four of them can be extracted into triples. Those attributes include:

- The *geometry* itself (*mandatory*). Utility expects valid shapes for points, (multi)linestrings, and (multi)polygons according to OGC simple features specification [OGC10b].
- A unique *identifier* (*mandatory*), which acts as a primary key for each entity and will be used for identifying the extracted resource.
- A *name* associated with an entity (*optional*). This value will be converted into a string literal.
- A *type* that characterizes an entity (*optional*). This value will be converted into a string literal.

Geometric data must reside in a *single* table (in case of a database) or a shapefile. Currently, there is no support for combining information from several sources (e.g., by joining two or more tables).

#### 3.2.2 Output data

In terms of output serializations, and according to the specifications of the Jena API [JenaDoc] that is used to export the model, the triples can be obtained in one of the following formats:

- **RDF/XML**. This is the *default* output format that represents RDF as XML, according to the RDF specifications. Note that an error may occur with this RDF/XML serialization in case of blank nodes in the model. Specifically, a blank node gets a URI reference in this format, and thus it is no longer
blank. So, the RDF/XML syntax is not capable of representing all RDF models; for example it cannot represent a blank node which is the object of two statements.

- **RDF/XML-ABBREV.** This syntax (called PrettyWriter by Jena API) takes advantage of features of the RDF/XML abbreviated syntax to write a Jena model more compactly. It is also able to preserve blank nodes where possible. However, it is not suitable for writing very large models, as its performance might not be acceptable for voluminous datasets.

- **N-TRIPLES.** This syntax is most preferable to write large files, and it also preserves blank nodes. However, it lacks some of the shortcuts provided by other RDF serialisations (e.g., N3, TTL).

- **N3.** Syntax Notation3 (or N3 as it is more commonly known) is a shorthand non-XML serialization of RDF models (not to be confused with N-TRIPLES syntax). N3 has been designed with human-readability in mind; hence, it is much more compact and readable than XML/RDF notation. N3 also offers features beyond a serialization for RDF models, such as support for RDF-based rules.

- **TURTLE** (also abbreviated as TTL). This syntax represents a Terse RDF Triple Language and provides a way to group three URIs to make a triple. It can also abbreviate such information, for example by factoring out common portions of URIs. Essentially, TURTLE is a simplified, RDF-only subset of N3.

In terms of standardization, the output triples are conformant to W3C standards, thanks to methods provided by the underlying Jena API for creating resources, properties and literals and the statements linking them. Therefore, all output triples are compatible with the most commonly used standards, including RDF, RDFS, OWL, and SPARQL.

With respect to geospatial features, triples can be exported according to the GeoSPARQL standard [OGC12]. In addition, **TripleGeo** offers the ability to export point geometries into custom namespaces for Virtuoso and WGS84 RDF Geoposition vocabulary [GeoPos84], but note that this syntax is not compliant to GeoSPARQL. Basically, the output format depends on the triple store where this data will be imported next:

- For uSeekM and Parliament, geometries must be compliant to GeoSPARQL.
- **Virtuoso** requires its own custom format for point features only.
- **OWLIM-SE** supports only points according to the WGS84 RDF vocabulary.
- Other stores (like Oracle RDF, Strabon, etc.) are close, but not conformant to GeoSPARQL, due to differing namespaces. In that case, geometries can be extracted in GeoSPARQL format and the resulting file can be edited for replacing the necessary prefixes.

More information about the differing geometric representation offered from various triple stores is available in Deliverable D2.1.1 [GeoKnowD21].

In the Appendix, we provide indicative examples of various types of geometries exported in several formats and geospatial representations using the **TripleGeo** utility.

### 3.3 Manual

#### 3.3.1 Installation instructions

Version 1.0 of **TripleGeo** is publicly available, offering the entire Java source code as well as a JAR package that contains executable binaries [TripleGeo]. For convenience, and in order to include all dependencies (cf. Section 3.1.2), as well as several configuration samples, it can be downloaded as a single .ZIP file that can be extracted into a separate folder in a local machine. Java JRE (or SDK) 1.7 (or later) must be installed and properly configured in order to execute **TripleGeo** from its binaries. **TripleGeo** has been successfully tested in both MS Windows and Linux environments.
3.3.2 Configuration file

Before attempting any conversion using TripleGeo, a configuration file must be prepared. This file lists crucial properties that define how input data will be accessed, where they will be exported and into which format, as well as optional features (e.g., reprojection into another spatial reference system).

Here is the general structure of this configuration file:

1. **Input and output parameters.** The user must specify (relative or absolute) paths for the working directory (parameter `tmpDir`), as well as the name of output file (`outputFile`). In case that input data comes in shapefile format, input file (`inputFile`) must be specified as well. This latter property should not be mentioned in case that data resides in a geospatial DBMS.
2. **Export format.** Possible export formats (defined under property `format`) include RDF/XML (default), RDF/XML-ABBREV, N-TRIPLES, TURTLE (or TTL), and N3.
3. **Target triple store.** The user should specify the type of the triple store where the exported data will be imported, since the geometric representation and geospatial support varies widely amongst them. Possible values for property `targetStore` include:
   i. `GeoSPARQL`, for subsequent import to compliant triple stores (e.g., Parliament, uSeekM),
   ii. `Virtuoso`, when extracting point features using the `virtrdf` namespace, and
   iii. `wgs84_pos`, for point features under the WGS84 Geoposition RDF vocabulary [GeoPos84].
4. **Database credentials and features.** These include all parameters necessary for connecting to a DBMS and extracting features from a spatial table. First, the user must specify the credentials that will be used for connecting to a spatial database (host IP address, database name, port, username, password). A specific DBMS platform must be defined with property `dbType` using one of these possible values: 1 (for MySQL), 2 (for Oracle Spatial 11g), 3 (for PostGIS), or 4 (for IBM DB2). Next, parameters for accessing features and attributes from a specific table may be defined (with case-insensitive values):
   i. `tableName` is mandatory and indicates the table that contains spatial features.
   ii. `resourceName` is mandatory and defines the name of the resources that will be created.
   iii. `condition` can be used to specify a filter for selecting specific qualifying records. This can be any valid condition(s), as if it were specified in the WHERE clause of an SQL statement (i.e., allowing use of AND, OR, LIKE, BETWEEN etc.). In case `condition` is left blank, all records of the table will be extracted.
   iv. `labelColumnName` is mandatory and specifies the attribute that uniquely characterizes the feature (i.e., the primary key of the table). This attribute value will be used as a common identifier for all triples derived from a single record.
   v. `nameColumnName` is optional and can be used to extract name literals.
   vi. `classColumnName` is optional and can be used to extract type literals.
   vii. `geometryColumnName` specifies the name of the geometry attribute (mandatory).
   viii. `ignore` can be used to specify values (e.g., UNK) in attributes that should not be exported as literals. By default, NULL values in attributes are suppressed and are not exported in order to avoid blank nodes.
5. **Shapefile features.** Should spatial data be extracted from a shapefile, the following values may be specified:
   i. `featureString` is mandatory and indicates and name of the spatial source file. This is the base name of the set of files that constitute a shapefile. For instance, if a shapefile consists of `{mypoints.shp, mypoints.shx, mypoints.dbf}`, then the value of this property is `mypoints` (i.e., without file extension).
   ii. `type` is mandatory and defines a user-defined name for the resources that will be created.
iii. attribute is mandatory and specifies the attribute that uniquely characterizes the feature (i.e., the primary key of the table). This is used as a common identifier for all triples derived from a single record.

iv. name is optional and can be used to extract name literals.

v. class is optional and can be used to extract type literals.

vi. ignore can be used to specify values (e.g., UNK) in attributes that should not be exported as literals. By default, NULL values in attributes are suppressed and are not exported in order to avoid blank nodes.

6. Namespace parameters. The namespaces and prefixes for the resources that will be generated as well as for the utilized ontology are set with these properties:

i. nsURI specifies the common URI namespace for all generated resources. In case this is not specified, a default value nsURI=http://geoknow.eu/resource/ is assumed.

ii. nsPrefix defines a prefix name for the utilized URI namespace (previously defined with property nsURI). If this value is not specified (i.e., an empty string in this configuration file), then nsPrefix=georesource is assumed.

iii. ontologyNS defines the namespace for the underlying geospatial ontology. Typically, the default value is ontologyNS=http://www.opengis.net/ont/geosparql# in order to generate features compliant to GeoSPARQL.

iv. ontologyNSPrefix defines a prefix name for the geospatial ontology (previously defined with property ontologyNS). If this value is not specified (i.e., an empty string in this configuration file), then a prefix ontologyNSPrefix=geontology is used.

7. Spatial Reference Systems. If geographic reprojection is required or wanted for the final triples, then properties sourceRS and targetRS must be defined by the user. TripleGeo works for any valid EPSG reference systems [EPSG] and transforms all geometries in the dataset, e.g., from sourceRS=EPSG:2100 (Greek Grid 1987) to targetRS=EPSG:4326 (WGS84).

8. Optional parameters. Basically, this affects the default language for any labels created in the output RDF. Unless otherwise specified, the default value is English-en (i.e., defaultLang=en).

Note that the URI of each generated feature and geometry comes from the concatenation of:

- the specified nsURI for the common namespace (e.g., http://geoknow.eu/resource/).
- the name of the resource defined either with property resourceName for DBMSs or property type for shapefiles (e.g., towns).
- the unique identifier of the original feature, as retrieved from the attribute specified with property labelColumnName for DBMSs or property attribute for shapefiles, respectively (e.g., an OSM feature with attribute value osm_id = 428).

For this example, the resulting resource of this feature in a triple will be like:

http://geoknow.eu/resource/towns_428

whereas its associated geometry will be identified in triples as:

http://geoknow.eu/resource/Geom_towns_428

by adding a Geom prefix before the name of this particular resource.

It should also be mentioned that utility geometry2Rdf [geo2rdf] makes use of additional properties in order to handle its custom geometric representations. For instance, URIs for Point, LineString and Polygon resources are defined, as well as a URI of relationship “formBy” (since a Linestring and a Polygon are “formBy” a sequence of Points). Such properties are entirely ignored by our TripleGeo utility during conversion, since the utilized WKT representations handle all supported types of geometries in a consistent manner.
3.3.3 Execution

In order to use TripleGeo for extracting triples from a spatial dataset, the user should follow these steps:

1. Open a terminal window and navigate to the directory where TripleGeo has been extracted. Normally, this folder includes a lib/ subdirectory with the required libraries, as well as a configuration file (e.g., named options.conf).
2. Verify that Java JRE (or SDK) version 1.7 or later is installed. Currently installed version of Java can be checked using: java -version from the command line.
3. Check all properties in the required configuration file, as explained in Section 3.3.2. If triples are to be extracted from a DBMS, make sure that the correct credentials are given in the configuration file. This file must be located in the same folder as the executable TripleGeo.jar package, assuming that Java binaries are bundled together in TripleGeo.jar. In case that you compile the utility directly from the Java source code, make sure that you provide the correct path to this configuration file.
4. In case that triples will be extracted from ESRI shapefiles, and assuming that Java binaries are bundled together in TripleGeo.jar, give the following command:
   
   ```
   java -cp lib/*;TripleGeo.jar eu.geoknow.athenarc.triplegeo.ShpToRdf options.conf
   ```

   Alternatively, if triples will be extracted from a geospatially-enabled DBMS (e.g., Oracle Spatial), give the following command:

   ```
   java -cp lib/*;TripleGeo.jar eu.geoknow.athenarc.triplegeo.wkt.RdbToRdf options.conf
   ```

5. While conversion is running, it periodically issues notifications about its progress. Note that for large datasets (i.e., hundreds of thousands of records), conversion may take several minutes.

   As soon as processing is finished and all triples are written into a file, the user is notified about the total amount of extracted triples and the overall execution time.

3.3.4 Known limitations

3.3.4.1 Handling large datasets

Judging from our experience with extraction of triples from several geospatial repositories (cf. Section 3.4.2), it seems that this process may take several minutes to conclude in case of datasets that include thousands of records. Hence, in case of extremely large datasets (e.g., millions of records), it is advisable to split the input in several smaller parts and then extract triples separately from each one. When large datasets are handled, execution settings should also include suitable values for Java heap size in main memory (i.e., calling the executable with the -Xms<size> option).

3.3.4.2 JDBC connection to geospatial DBMSs

Connection to a geospatial DBMS is performed through a JDBC bridge, so a suitable driver should be available. TripleGeo is shipped with several such freely available drivers (e.g., for PostgreSQL), but certain software vendors restrain usage of such tools only to customers that have purchased their DBMS platform (e.g., IBM DB2). So, before attempting to execute TripleGeo against data residing in such platforms, the user should make sure that the necessary JDBC driver(s) for that version of the DBMS software are installed in subdirectory lib/ and thus accessible by the TripleGeo utility.
3.3.4.3 Interacting with Oracle 11g databases on Linux platforms

When attempting to export triples from Oracle Server Enterprise Edition - Version: 11.1.0.6 to 11.2.0.2.0 [Release: 11.1 to 11.2] on Linux platforms, connection is established via the suitable JDBC driver. But as soon as records are to be retrieved, the following error may be issued from Oracle:

ORA-29516: Error in module Aurora: Assertion failure at joez.c:3311
Bulk load of method java/lang/Object.<init> failed; insufficient shm-object space

It seems that this error has to do with the just-in-time (JIT) compiler for Oracle JVM environment, which is intended for faster execution as invalidation, recompilation, and storage of code is done dynamically. JIT is controlled by parameter java_jit_enabled, and if it is set to TRUE then the Java methods are automatically compiled to native code by the JIT compiler and made available for use by all sessions.

But if error ORA-29516 appears on a Linux x64bit platform, the workaround to overcome that error is to turn off the JIT compiler by giving this SQL command to Oracle (administrative privileges are required):

ALTER SYSTEM SET java_jit_enabled=false;

Afterwards, exporting of triples is carried out without errors, but at the expense of a rather slow rate especially for larger datasets, as indicated from evaluation results in Section 3.4.

3.4 Evaluation

3.4.1 Use case

In order to prepare the Market and Research Overview for Deliverable D2.1.1 of this project [GeoKnowD21], it was necessary to install and evaluate the functionality and geospatial support of several triple stores. Hence, we had to populate each one with geographic features and conduct an evaluation study for assessing their efficiency. Therefore, we were able to test TripleGeo with several input formats and geometric representations for the output triples. Not only has this task proven the robustness of TripleGeo, but the differing geospatial specifications of each triple store also guided its development and progressive refinement towards handling as many cases as possible.

In our test case scenario, OpenStreetMap [OSM] data for Great Britain covering England, Scotland, and Wales as of 05/03/2013, were downloaded in ESRI shapefile format [OSM_GB]. Of the available OSM layers, only those concerning road network (roads), points of interest (points) and natural parks and waterbodies (natural) were actually utilized. These layers were chosen as representatives for each geometry type (respectively points, linestrings, and polygons). They were deemed most meaningful for queries involving multiple layers, and they also contained many more features compared to other similar layers. From each original layer, only the most important attributes were retained, as detailed in Table 1.

<table>
<thead>
<tr>
<th>OSM Layer</th>
<th>Geometry type</th>
<th>Description</th>
<th>Attributes examined</th>
<th>File size (MBytes)</th>
<th>Number of features</th>
</tr>
</thead>
<tbody>
<tr>
<td>Points</td>
<td>Point</td>
<td>Points of general interest (bus stop, pub, hotel, traffic signal, place of worship, etc.)</td>
<td>shape, name, osm_id, type</td>
<td>74.3</td>
<td>590390</td>
</tr>
<tr>
<td>Roads</td>
<td>LineString</td>
<td>Road network characterized according to OSM</td>
<td>shape, name, osm_id, type</td>
<td>706</td>
<td>2601040</td>
</tr>
</tbody>
</table>
(spaghetti) classification (motorway, trunk, primary, secondary, tertiary, residential etc.)

<table>
<thead>
<tr>
<th>Natural</th>
<th>Polygon</th>
<th>Surfaces like parks, forests, waterbodies (lakes, riverbanks etc.)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>shape, name, osm_id, type</td>
</tr>
<tr>
<td></td>
<td></td>
<td>152 264570</td>
</tr>
</tbody>
</table>

All data was used “as is”, without modifying existing geometric or descriptive information. Features were already georeferenced in WGS84, so no transformation was necessary. In terms of descriptive information, the unique OSM identifier was kept as the reference label for each feature, whereas attributes for name and type (if not null), were also extracted as string literals.

Original shapefiles were converted into triples with **TripleGeo**, thus providing the datasets used for examining the state-of-the-art in geospatial support of several triple stores (i.e., Virtuoso, OWLIM-SE, uSeekM, Strabon, and Parliament, as detailed in Deliverable D2.1.1 [GeoKnowD21]). However, these shapefiles were also imported into PostGIS and Oracle Spatial 11g databases. Thus, we have also conducted export operations from these spatial repositories into triples, and we were able to verify that **TripleGeo** can also interact and access spatial features from major DBMSs. Next, we report some indicative experimental results from such conversions.

### 3.4.2 Experimental validation

Utility **TripleGeo** has been implemented for integration with several geospatial repositories. It has been successfully tested and provably works for data stored in these platforms:

- ESRI shapefiles (in both MS Windows and Linux environments);
- Oracle Spatial 11g (in both MS Windows and Linux environments);
- PostgreSQL 9.1 with PostGIS 1.5.3 (in Linux);
- PostgreSQL 9.2 with PostGIS 2.0 (in MS Windows);
- MySQL 5.6 (in MS Windows); and
- IBM DB2 9.5 with Spatial Extender (in MS Windows).

Apart from verification of its functionality, we have also performed some more comprehensive tests for converting larger datasets into triples. For these experiments, we have setup a XEN hypervisor on an evaluation Linux server (Intel Core i7-3820 CPU at 3.60GHz with 10240KB cache), which is capable of hosting a group of Virtual Machines (guest VMs). Each VM used in the experiments was given 8GB RAM, 2GB swap space, 4 (virtual) CPU cores and 40GB disk of storage space. During each experiment, only the VM being evaluated was active.

Using **TripleGeo**, we performed conversions from geospatial OSM data of the use case presented in Section 3.4.1. The same OSM datasets for Great Britain were stored in the following platforms:

- ESRI shapefiles;
- Oracle 11gR2 (incl. Spatial) Standard Edition 11.2.0.1.0;
- PostgreSQL 9.1 with PostGIS 1.5.3.

Conversion into triples has taken place for most typical geometric representations in GeoSPARQL (concerning points, linestrings, and polygons), Virtuoso RDF (points only) and WGS84 Geoposition RDF Vocabulary (points only). Table 2 indicates the amount of triples extracted from the respective OSM layers for each representation. Observe that the number of triples resulting from the point dataset varies among the supported RDF syntaxes. As can be verified from the example triples in the Appendix, Virtuoso RDF...
provides a minimal representation for points with the geometric literal in a single statement. In contrast, WGS84 Geoposition RDF Vocabulary includes two statements per point in order to properly define its lat/long coordinates [GeoPos84]. Finally, GeoSPARQL is more verbose, as each entity should be characterized as a GeoSPARQL feature, whereas it must be also associated with its geometry and its geometry type, each one requiring a separate statement.

Table 2: Data contents of OSM layers (ESRI shapefiles)

<table>
<thead>
<tr>
<th>OSM Layer</th>
<th>Geometry type</th>
<th>GeoSPARQL</th>
<th>Virtuoso RDF</th>
<th>WGS84 RDF Vocabulary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Points</td>
<td>Point</td>
<td>4446368 triples</td>
<td>2675212 triples</td>
<td>3265602 triples</td>
</tr>
<tr>
<td>Roads</td>
<td>LineString</td>
<td>19472612 triples</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Natural</td>
<td>Polygon</td>
<td>1877672 triples</td>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>

In addition, we measured how much time TripleGeo required for: (i) parsing initial records and turn them into triples in a Jena RDF model, and (ii) exporting this model as a file in a suitable format (e.g., RDF/XML, TTL, N3 etc.). Table 3 reports indicative execution times for the three platforms (ESRIshapefiles, PostGIS, and Oracle Spatial) and the supported geometric RDF representations. Quite expectedly, processing takes a few minutes for layer points of this specific OSM dataset, no matter the geometric RDF representation. The same remark also holds for the polygon features (layer natural), but in that case the reason must be the relatively small number of features. In contrast, exporting millions of linestrings (layer roads) takes much more time (about two hours and a half, at best) and varies a lot among platforms. Such delays should be mostly attributed to memory shortage, as the entire RDF model is retained in main memory and grows proportionally to the amount of statements generated per initial record. It must also be stressed that interaction with Oracle Spatial was rather slow due to deactivation of its JIT compiler, as mentioned in Section 3.3.4. Presumably, the process crashed when exporting triples into file, as the system runs out entirely from memory resources. This case signifies that triple extraction for large datasets with millions of features should better be performed in several smaller batches of the original data.

Table 3: Processing time when extracting triples from various geospatial repositories

<table>
<thead>
<tr>
<th>ESRI Shapefiles</th>
<th>GeoSPARQL</th>
<th>Virtuoso RDF</th>
<th>WGS84 RDF Vocabulary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Points</td>
<td>parsing</td>
<td>exporting</td>
<td>parsing</td>
</tr>
<tr>
<td>Roads</td>
<td>155159 ms</td>
<td>129062 ms</td>
<td>168570 ms</td>
</tr>
<tr>
<td>Natural</td>
<td>218508 ms</td>
<td>67392 ms</td>
<td>N/A</td>
</tr>
<tr>
<td>PostGIS Points</td>
<td>parsing</td>
<td>exporting</td>
<td>parsing</td>
</tr>
<tr>
<td>Roads</td>
<td>5851172 ms</td>
<td>2920093 ms</td>
<td>N/A</td>
</tr>
<tr>
<td>Natural</td>
<td>218508 ms</td>
<td>67392 ms</td>
<td>N/A</td>
</tr>
<tr>
<td>Oracle Spatial</td>
<td>parsing</td>
<td>exporting</td>
<td>parsing</td>
</tr>
<tr>
<td>Points</td>
<td>117332 ms</td>
<td>128346 ms</td>
<td>110007 ms</td>
</tr>
<tr>
<td>Roads</td>
<td>4138529 ms</td>
<td>5268280 ms</td>
<td>N/A</td>
</tr>
<tr>
<td>Natural</td>
<td>77406 ms</td>
<td>64649 ms</td>
<td>N/A</td>
</tr>
</tbody>
</table>

3.5 Future work

Utility TripleGeo already provides a considerable level of integration with geospatial databases and files. However, more functionality can be added without affecting the existing code, thanks to its modular implementation. Among the features planned for updated versions are:
• Ability to extract geometries from other widely used file formats like: GML, KML, MapInfo MID/MIF or TAB formats, AutoCAD DXF etc. Access to most such file formats is already implemented by GeoTools [GeoTools] and OGR/GDAL libraries [GDAL].
• Interaction with more DBMS platforms that offer geospatial support, such as MS SQL Server [MS-SQL-Server], Informix [Informix] etc. This requires formulation of the necessary SQL queries and extraction of WKT representations for the returned geometries.
• Support for additional geometric types according to OGC specifications [OGC10b]. Existing functionality covers the most commonly used types: points, linestrings and multilinestrings, as well as polygons and multipolygons. However, it would be beneficial to support more geometric shapes such as multipoints, geometry collections, etc.
• Ability to export additional non-spatial attributes as literals. Note that existing functionality covers three such attributes (an identifier, a name and a type for each feature) at maximum.
• Optimize performance when dealing with large datasets. With respect to writing the output file, a possible solution would be to automatically split the resulting triples into separate batches (e.g., a series of files of one million triples each).

3.6 Licensing

The TripleGeo tool is free software and its current version (including the Java source code and sample data) is available from [TripleGeo]. It can be redistributed and/or modified under the terms of the GNU General Public License as published by the Free Software Foundation; either version 3.0 of the License, or (optionally) any later version. A copy of the GNU General Public License should have been received along with this tool.

This tool is distributed in the hope that it will be useful, but without any warranty; without even the implied warranty of merchantability or fitness for a particular purpose. Please consult the GNU Lesser General Public License for more details [GPL3].
4. Conclusions

In this deliverable we presented tools that we have developed for importing data from external geospatial repositories into triple stores. The first tool, **Sparqlify**, and is a novel SPARQL-to-SQL query rewriter, which enables SPARQL queries on relational databases and simplifies definition of RDF views. Such mappings are expressed in SPARQL grammar extended with a few custom production rules. After validating Sparqlify against LOD benchmarks, it turns out that it is scalable for large datasets, which is crucial when mapping large crowdsourced datasets. Besides, a second tool offers the means to make import of diverse standards-compliant and diversely modelled geodata as much automatic as possible. That utility is called **TripleGeo** and can integrate geometries and attributes directly from several geospatial databases or widely used GIS formats. Not only can this data be converted into triples, but this process also includes functionality for on-the-fly coordinate transformations, as well as exporting into diverse serializations. Both tools are already publicly available, and they have been successfully showcased against OpenStreetMap datasets, thus offering concrete evidence about their robustness and suitability for importing large geospatial datasets.
5. References


[BSBMmap] Available at http://wifo5-03.informatik.uni-mannheim.de/bizer/berlinsparqlbenchmark/v2/results/store_config_files/d2r-mapping.n3

[BSBMres] BSBM results document. Available at http://wifo5-03.informatik.uni-mannheim.de/bizer/berlinsparqlbenchmark/results/index.html#d2r

[DATAlinc] Open Database License. Available at http://opendatacommons.org/licenses/odbl/1.0/


[D2RServer] D2R Server: Accessing databases with SPARQL and as Linked Data. Available at http://d2rq.org/d2r-server


[EXBG] Example boundary of Germany. Available at http://linkedgeodata.org/page/triplify/rmembers51477


[GeoPos84] Basic Geo (WGS84 lat/long) Vocabulary. Available from http://www.w3.org/2003/01/geo/

[GLD] The GeoLinked Data (.es) team. Enrich the Web of Data with Spanish geospatial data. Available at http://geo.linkeddata.es/


[JERS] Jersey. Available at https://jersey.java.net/


[LGD] AKSW Team. LinkedGeoData project: Adding a spatial dimension to the Web of Data. Available at http://linkedgeo-data.org


[LGDgit] LinkedGeoData at github. Available at https://github.com/GeoKnow/LinkedGeoData

[LGDLic] LinkedGeoData GPL v3 License. Available at https://github.com/GeoKnow/LinkedGeoData/blob/master/LICENSE

[LOD2] LOD2 project: Creating Knowledge out of Interlinked Data. Available at http://lod2.eu


[map4rdf] GeoLinkedData Team. map4rdf: Maps viewer of RDF resources with geometrical information. Available at http://oeg-dev.dia.fi.upm.es/map4rdf/


[OSM] OpenStreetMap project. Available at http://www.openstreetmap.org/


[PING] Pingback 1.0. Available at http://www.hixie.ch/specs/pingback/pingback


[PostGIS] PostGIS - Spatial and Geographic objects for PostgreSQL. Available at http://postgis.net/


[PUB] Original Pubby project. Available at https://github.com/cygri/pubby

[PUBspar] Adapted version of Pubby for Sparqlify. Available at https://github.com/Aklakan/pubby


[R2RML] R2RML and Direct Mapping Test Cases. Available at http://www.w3.org/TR/r2rml/


[R2RML] R2RML: RDB to RDF Mapping Language. Available at http://www.w3.org/TR/r2rml/


[SLFJ] Simple Logging Facade for Java. Available at http://www.slf4j.org/

[SNORQL] SNORQL. Available at https://github.com/kurtjx/SNORQL


[SPARAnal] SparqlAnalytics. Available at https://github.com/AKSW/SparqlAnalytics


[SpatialETL] Wikipedia article on Spatial ETL. Available at http://en.wikipedia.org/wiki/Spatial_ETL

[SPRQLF] Sparqlify. Available at https://github.com/AKSW/Sparqlify


[SPPRQLFgit] Sparqlify at github. Available at https://github.com/AKSW/Sparqlify


[TRWI] TranslateWiki. Available at http://translatewiki.net/

[Ultrawrap] Ultrawrap. Available at http://ribs.csres.utexas.edu/ultrawrap/


6. Appendix

Next, we provide an example of triples extracted by utility TripleGeo from OSM datasets for Great Britain. For a given record concerning a point of interest (POI) with OSM_ID=105, the following statements provide information about its resource class (arbitrarily named uk_points), as well as three non-spatial attributes: its identifier (105), name ("The Green Man") and POI type ("pub"), all of them formalized as string literals:

- **Statements concerning non-spatial attributes (4 triples)**

  ```
  <http://geoknow.eu/uk_points#points_105> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://geoknow.eu/uk_points#points> .


  <http://geoknow.eu/uk_points#points_105> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://geoknow.eu/uk_points#pub> .
  ```

With respect to its geometry, the resulting RDF representation differs, depending on the specification supported by the triple store where the data will be imported. Utility TripleGeo supports three alternative geometric representations, each with its own syntax:

- **Geometric representation according to GeoSPARQL specification (4 triples)**

  ```


  <http://geoknow.eu/uk_points#Geom_points_105> <http://www.opengis.net/ont/geosparql#asWKT> "POINT(\-0.1430911 51.5238028)"^^<http://www.opengis.net/ont/geosparql#wktLiteral> .
  ```

- **Geometric representation according to Virtuoso RDF Vocabulary (1 triple)**

  ```
  ```

- **Geometric representation according to WGS84 GeoPositioning RDF Vocabulary (2 triples)**

  ```
  <http://geoknow.eu/uk_points#points_105> <http://www.w3.org/2003/01/geo/wgs84_pos#long> \-0.1430911 .

  <http://geoknow.eu/uk_points#points_105> <http://www.w3.org/2003/01/geo/wgs84_pos#lat> 51.5238028 .
  ```
Namespaces typically used as RDF Prefixes in conversions

geo: <http://www.opengis.net/ont/geosparql#>  OGC GeoSPARQL datatypes & relational properties
owl: <http://www.w3.org/2002/07/owl#>
pos: <http://www.w3.org/2003/01/geo/wgs84_pos#>  WGS84 Geo Positioning: an RDF vocabulary
rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
rdfs: <http://www.w3.org/2000/01/rdf-schema#>
sf: <http://www.opengis.net/ont/sf#>  OGC GeoSPARQL spatial functions
virtrdf: <http://www.openlinksw.com/schemas/virtrdf#>  Virtuoso’s prefix for RDF data
xsd: <http://www.w3.org/2001/XMLSchema#>

User-defined namespaces in the tested datasets

poi: <http://geoknow.eu/uk_points#>  Points of interest (point)
roads: <http://geoknow.eu/uk_roads#>  Road network (linestring)
zones: <http://geoknow.eu/uk_natural#>  Natural parks and waterbodies (polygon)