DELIVERABLE D2.2

Mapping specification and POI transformation service

SLIPO

PROJECT NUMBER: 731581
START DATE OF PROJECT: 01/01/2017
DURATION: 36 months

Horizon 2020

SLIPO is a research project funded by European Commission's Horizon 2020 Programme.

Dissemination Level	Public
Due Date of Deliverable	Month 15, 30/03/2018
Actual Submission Date	29/03/2018
Work Package	WP2 - POI Modelling, Identifiers, and Transformation
Tasks	2.3 Transforming POIs as Linked Data
Type	Other
Approval Status	Submitted for approval
Version	1.0
Number of Pages	101
Filename	D2.2_Mapping_specification_and_POI_transformation_service.pdf

Abstract

This document presents initial version of the POI transformation software of the SLIPO Toolkit. First, we provide an overview of the transformation process as employed during the POI data integration lifecycle in SLIPO, and we survey the current state-of-the-art in geospatial data transformation to RDF. Then, we give an overview of the initial, current, and envisaged full-fledged functionality of TripleGeo, the transformation software of the SLIPO Toolkit. We thoroughly analyse its components and explain its current support for attribute mappings and classification schemes for POIs. We also describe its novel functionality for reverse transformation from RDF to de facto geospatial formats. Finally, we report performance results from a comprehensive evaluation of the software that confirms its efficiency against a variety of input POI representations and formats, underscoring its potential for coping with scalable data volumes.

History

version	date	reason	revised by
0.1	30/01/2018	Contents and structure, abstract	Giorgos Giannopoulos, Spiros Athanasiou
0.2	22/02/2018	Description of TripleGeo v1.1	Kostas Patroumpas
0.3	28/02/2018	New versions of TripleGeo	Kostas Patroumpas
0.4	14/03/2018	Architecture & components	Kostas Patroumpas, Axel Ngonga, Mohamed Sherif
0.5	19/03/2018	Added experimental results	Kostas Patroumpas, Spiros Athanasiou
0.6	21/03/2018	Added description of mappings	Kostas Patroumpas, Giorgos Giannopoulos
0.7	25/03/2018	Text restructuring; added Annex	Kostas Patroumpas, Spiros Athanasiou
0.8	27/03/2018	Description of Identifiers Registry; Refined introduction & conclusion	Dimitrios Skoutas, Kostas Patroumpas, Claus Stadler
0.9	28/03/2018	Internal review	Dimitrios Skoutas, Kostas Patroumpas, Claus Stadler
1.0	29/03/2018	Final version	Spiros Athanasiou

Author list

organization	name	contact information
ATHENA RC	Spiros Athanasiou	spathan@imis.athena-innovation.gr
ATHENA RC	Giorgos Giannopoulos	giann@imis.athena-innovation.gr
ATHENA RC	Kostas Patroumpas	kpatro@imis.athena-innovation.gr
ATHENA RC	Dimitrios Skoutas	dskoutas@imis.athena-innovation.gr
InfAI	Axel Ngonga	ngonga@informatik.uni-leipzig.de
InfAI	Mohamed Sherif	sherif@informatik.uni-leipzig.de
InfAI	Claus Stadler	cstadler@informatik.uni-leipzig.de

Executive Summary

This document presents initial version of the POI transformation software of the SLIPO Toolkit, which is the *entry point* for POI datasets in the SLIPO lifecycle. Such data assets may come from a variety of spatial repositories (*databases, geographical files, Web APIs, etc.*) and may have been maintained under diverse schema representations. Hence, Extract-Transform-Load (ETL) processes, tools, protocols, and workflows are required for mapping and transforming such POI data and metadata into RDF adhering to the common, vendor-agnostic, and extendable SLIPO POI ontology presented in Deliverable D2.1.

TripleGeo implements interfaces for accessing POI data in diverse geospatial file formats and DBMSs and extracting them into RDF with geometries under the OGC GeoSPARQL standard. It also incorporates facilities that enable their mapping to an ontology according to well-established vocabularies, as well as assignment of POI entities to categories under a user-specified classification scheme. Since POI data need to be accessible and exploitable by existing software and services, TripleGeo also includes facilities for the *reverse transformation* of RDF triples into POI formats. Compared to its initial release, its current version 1.4 has improved support for a great variety of input POI representations and formats, can handle all thematic (i.e., non-spatial) attributes available per POI, and is able to transform millions of POIs in a few minutes. This testifies the potential of this software to cope with even more scalable POI datasets in future releases, by employing more efficient data partitioning schemes and parallelization in modern cluster infrastructures.

The layout of this document is as follows.

In Section 1, we introduce the setting of the transformation task in the context of SLIPO. We describe the objectives of this task and offer a survey of related work on transformation of geospatial information to RDF. Finally, we briefly report our achievements regarding the development of TripleGeo tool until M15, and the features currently available in its ver.1.4.

In Section 2, we discuss our progress in developing the TripleGeo software, starting from its original source stack (ver.1.1) obtained from the GeoKnow project, describing the improved functionality available in the current release (ver.1.4), and outlining our plans for advanced efficiency and scalability for the final release (ver.2.0).

In Section 3, we provide a detailed account of the transformation process as carried out by the current version of TripleGeo. This includes support for attribute mappings and classification schemes, as well as an API for registering POIs in the SLIPO Identifiers Registry. We also discuss our initial scheme for achieving scalability over large geospatial datasets with multiple worker threads.

In Section 4, we outline the processing flow of reverse transformation and explain how it is possible to reconstruct RDF data of POIs with geometries into records stored in a geospatial file.

In Section 5, we present a complete user's guide to TripleGeo ver.1.4 along with indicative execution examples that demonstrate its operation against POI datasets.

In Section 6, we evaluate the efficiency and scalability of TripleGeo against large POI datasets extracted from OpenStreetMap and stored in various geospatial repositories. This experimental evaluation confirms

the versatility of the software and testifies its potential for handling even larger POI datasets in future releases.

Section 7 offers conclusions and outlines our plans for future extensions of the TripleGeo software.

Finally, in the Annex we provide indicative examples on configuring TripleGeo execution, as well as sample mappings and classification schemes applied on POI datasets.

Abbreviations and Acronyms

API	Application Programming Interface
CRS	Coordinate Reference System
CSV	Comma Separated Values
DBMS	DataBase Management System
EPSG	European Petroleum Survey Group
ETL	Extract-Transform-Load
GIS	Geographical Information Systems
GML	Geography Markup Language
GPS	Global Positioning System
GPX	GPS Exchange Format
GUI	Graphical User Interface
HDFS	Hadoop Distributed File System
INSPIRE	INfrastructure for SPatial InfoRmation in Europe
ISO	International Organization for Standardization
JDBC	Java DataBase Connectivity
JSON	JavaScript Object Notation
KML	Keyhole Markup Language
MBR	Minimum Bounding Rectangle
OGC	Open Geospatial Consortium
OSM	OpenStreetMap
OWL	Web Ontology Language
POI	Point of Interest
RDF	Resource Description Framework
RDFS	Resource Description Framework Schema
REST	REpresentational State Transfer
RML	RDF Mapping Language
SPARQL	SPARQL Protocol and RDF Query Language

SQL	Structured Query Language
UML	Unified Modelling Language
URI	Uniform Resource Identifier
UUID	Universally Unique IDentifier
VM	Virtual Machine
WFS	Web Feature Service Interface (OGC Standard)
WMS	Web Map Service (OGC Standard)
WKT	Well Known Text
WGS84	World Geodetic System 1984 (EPSG:4326)
W3C	World Wide Web Consortium
XML	Extensible Markup Language
XSD	XML Schema Definition
XSL	EXtensible Stylesheet Language
XSLT	Extensible Stylesheet Language Transformation
YAML	YAML Ain't Markup Language

Table of Contents

1. Introduction

In this Section, we first define the transformation task and its goals regarding the integration of Points of Interest (POI) data. Then, we present the current state of the art in geospatial representation and transformation to RDF. Finally, we briefly outline the achievements of the current version 1.4 of TripleGeo software, which provides the main transformation utility in SLIPO.

1.1. POI Data Transformation

1.1.1. Task Definition

Considering the POI integration lifecycle illustrated in Figure 1, the transformation process in SLIPO Task 2.3 is a prerequisite for enabling processing against POIs as Linked Data. Indeed, transformation turns conventional POI and third-party datasets into RDF, thus enabling us to address the POI data integration challenge using the most effective technologies for this task: linked data. The data sources may be DBMSs, files, Web APIs, etc. which may provide a wealth of POI data assets, possibly employing diverse geometry representations, different attribute schemata, and possibly assigning categories to POIs under varying classification schemes. However, by transforming original POI data into a RDF representation according to a consistent, extensible, and rich ontology, all tasks involved in POI data integration (i.e., interlinking, fusion, quality assurance, etc.) are applicable in the Linked Data domain. Once transformation to RDF complete, POI features can be interlinked, fused, and enriched in successive steps that increase the size and the quality of the POI data in a virtuous cycle that implements an iterative workflow as shown in Figure 1.

However, it should be stressed that transformation is actually a *two-way process* that also allows the *reverse transformation* of linked POI data into conventional formats (i.e. de facto POI formats), thus enabling existing products, systems, and services to exploit the integrated POI datasets.

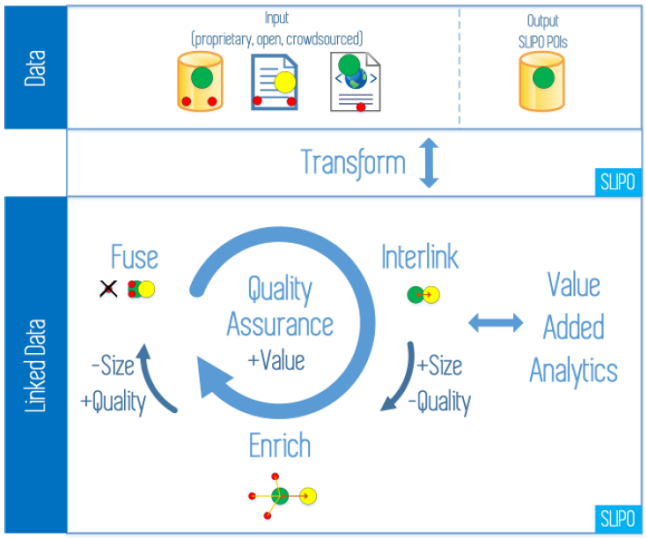


Figure 1: POI integration lifecycle

1.1.2. Importance and Challenges

The primary pillar and underlying idea of SLIPO is to address the POI data integration challenge by applying Linked Data technologies, which are ideally suited to handle the inherent geospatial, thematic, and semantic ambiguities of POIs. Hence, the entry point for POI datasets in the SLIPO lifecycle comprises transforming POI and POI-related Big Data assets into RDF data conforming to common and extendable schemas. This includes all necessary *Extract-Transform-Load* (ETL) processes, tools, protocols, and workflows for transforming POI data and metadata from diverse and disconnected data sources, schemas and formats, to RDF data adhering to a common, vendor-agnostic, well defined, yet agile and extendable SLIPO POI ontology [SLIPO-D2.1].

Data sources can be distinguished into two main categories: (a) POI entity descriptions and metadata, and (b) auxiliary data that complement, enrich, and contextualize POIs. The first category includes datasets represented by a series of different POI formats, each one carrying its own attribute schema. To handle these datasets, schema mappings that map concepts and attributes of the individual, conventional schemas to classes and properties of the SLIPO POI ontology are applied through data transformation tools. These tools implement interfaces and parsers for extracting POI data from diverse format sources, including specific format files, DBMSs, Web APIs, etc. To handle arbitrary schemas found in data under the second category, the transformation tools incorporate facilities for manual, semi-automatic and automatic schema mapping. To support this functionality, several well-established vocabularies are incorporated so that the mapping process utilizes standardized concepts and terms in the POI ontology.

After the transformations are applied, the resulting RDF triples are stored in standalone or distributed RDF stores, depending on their volume and volatility, providing access in both human and machine-readable form through SPARQL endpoints. As such, they can be used under the Linked Data paradigm for interlinking, fusion, enrichment, and analytics. Finally, since POI data need to be accessible and exploitable by existing software and services, transformation also includes facilities for the reverse transformation of RDF triples into any of the major POI formats.

1.1.3. Transformation Goals in SLIPO

In SLIPO, we proposed a comprehensive and vendor-agnostic ontology for POI data [SLIPO-D2.1], allowing it to model and represent multifaceted and enriched POI profiles, including temporal and evolving information. This accommodates and extends existing POI formats, providing a uniform and semantically rich model for assembling and managing POI data from heterogeneous sources.

Based on this model, we develop processes for creating persistent, unique, vendor and technology independent URIs for POIs exposed as Linked Data. In addition, we created a directory of identifiers (*SLIPO Identifiers Registry*), where all identifiers will be registered when first created and associated with key metadata about the respective POI.

In Task 2.3, we develop reusable processes and tools employing ontology mapping techniques, which are able to transform POI data from various types of sources (geographical vector files, CSV files, relational databases, semi-structured files like JSON or XML, etc.) to their RDF representations according to the POI data model. The implemented software can handle several POI transformation issues, such as scaling and efficiency, POI updating, schema mapping, and diversity of input formats and schemas. It also supports and

integrates mappings between the schemas of existing POI formats to the POI ontology established in SLIPO. Through these transformations, it will be possible to import POI data from various sources, including existing systems and products, transfer and address the data integration challenges in the Linked Data domain. In addition, we introduce a reverse transformation utility that can export the results of SLIPO data integration back to conventional geospatial representations.

Finally, we emphasize on the efficient and scalable implementation of the transformation software, aiming to support fast transformation of very large data collections of POIs. The design and development of the software should apply provably scalable paradigms, in particular cloud technologies for elasticity and fault tolerance. Since the beginning of the project, we have introduced constant benchmarking and evaluation experiments regarding performance with large POI datasets, and we incrementally inflate data in size and complexity (not only points, but more detailed geometries, as well as more attributes) in scalability tests. Our target for M36 is to become able to transform 100 Million POIs within a minute. To this end, we will consider data partitioning and parallelization options, both in transformation processing and in RDF storage.

1.2. Current State-of-the-Art

Creating knowledge from structured (e.g., relational databases, XML) or unstructured sources (e.g., text, images) can be extremely valuable in the Semantic Web. The R2RML Recommendation [R2RML] by W3C specifies an RDF notation for mapping relational tables, views or queries into the RDF data model. An alternative W3C recommendation concerns a direct mapping of relational data to RDF [RDFDirect] defines a simple transformation that can be used to materialize RDF graphs or define virtual graphs, which can be queried by SPARQL. Besides, the more generic RDF Mapping Language RML [DSC+14],[DKF+15] is defined as a superset of R2RML and aims to support not only relational databases, but also other data sources (CSV, XML, or JSON formats). RML can express customized mapping rules from heterogeneous data structures and serializations to RDF.

A variety of tools and scripts [ConverterToRdf] have been proposed for transforming data from an application-specific format into RDF for use with RDF tools and integration with other data, most of them considered as proof-of-concept prototypes [UHAS12]. Some of them are rather "mature" tools for transforming relational databases into RDF, such as Triplify [ADL+09], D2R Server [D2RServer], or Virtuoso's Sponger [Sponger]. During conversion, these tools allow reuse of existing vocabularies and ontologies. However, these tools lack support for geospatial data and operations.

On the other hand, several *Extract-Transform-Load (ETL) tools* can manage the unique characteristics of spatial data. Among them, GDAL/OGR [GDAL] is an open-source translator library implementing the OGC vector model [OGC10] and can handle proprietary storage models for many geospatial DBMSs and files. GeoKettle [GeoKettle] is a metadata-driven spatial ETL tool dedicated to integration of different data sources for building and updating geospatial data warehouses. Finally, FME Workbench is included in ESRI's ArcGIS Data Interoperability extension [FMEworkbench] and enables transformation of geometric and thematic attributes along with schema redefinitions. Currently, such utilities are mainly used for data cleaning, merging, verification or conversion into various formats, but have absolutely no RDF support.

There have been several proposals for geospatial RDF data management such as [GeoJSON],[GeoRDF], [GeoPos84], but none provided a solid framework for developing large-scale applications and services.

Since 2012, the OGC GeoSPARQL standard [OGC12] suggests a concrete ontology for representing features and geometries in RDF as Well Known Text (WKT) or Geography Markup Language (GML) literals. GeoSPARQL defines a core set of classes, properties and data types that can be used to construct query patterns in an extension of SPARQL. To cope with incompatible methods for representing and querying spatial data, GeoSPARQL follows other OGC standards [OGC10]. With such standardization, both vendors and users can achieve uniform, transparent, platform-independent access to geospatial RDF data with a rich collection of query operators. RDF stores have been increasingly adhering to implementations of the GeoSPARQL standard (e.g., Oracle Spatial and Graph [Oracle], Parliament [Parliament], or Virtuoso [Virtuoso]), but still some triple stores still maintain their own proprietary geometric representations (e.g., AllegroGraph [AllegroGraph]).

To the best of our knowledge, there have been very few initiatives specifically for converting *geospatial features into RDF resources*. *Geometry2RDF* [geo2rdf] enables geospatial data conversion from different formats (ESRI Shapefile, GML and geospatial DBMS) into RDF [VVS+10] according to the *NeoGeo vocabulary* [NeoGeo] and is not compliant with the GeoSPARQL standard [OGC12]. Data conversion into an appropriate RDF format using a selected ontology is among the functionalities supported by the generic *DataLift* platform [DataLift]. Although initially geometries could be extracted as WKT strings under a custom namespace in order to be queryable with SPARQL, support for GeoSPARQL-compliant geometries was included with its module *GeomRDF* [HABF14]. By default, GeomRDF generates predicates by reusing attribute names, which can be replaced afterwards by predicates from a vocabulary using another module using another Datalift module. In a different approach, *LinkedGeoData* [LGD] aims at adding a spatial dimension to the Semantic Web. It offers a flexible platform for mapping OpenStreetMap (OSM) data [OSM] 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. The resulting graph comprises billions of triples interlinked with DBpedia [DBpedia] and GeoNames [GeoNames]. Nevertheless, spatial operations deal strictly with OSM nodes and ways, ignoring any other geographic sources or data types. Specifically for Oracle Spatial and Graph, custom *RDF views* [OracleRDFViews] on relational data can be created via SQL queries or use of non-standard mappings (either with a direct mapping or with an R2RML mapping document) and physically store the generated triples in an RDF model within Oracle. As Oracle supports GeoSPARQL vocabularies, this data is also queryable with topological operations and spatial functions.

GeoTriples [KVS+14] extends mapping languages R2RML and RML with new constructs specifically for transforming geospatial entities into RDF. Although not adhering to a specific geospatial vocabulary, it supports the GeoSPARQL standard. It automatically generates and processes such mappings from diverse geospatial formats including relational databases (PostGIS and MonetDB), ESRI shapefiles, XML, GML, KML, JSON and GeoJSON documents and CSV documents. In that respect, GeoTriples seems share many of the capabilities of our transformation tool TripleGeo. However, it lacks support for as many input formats as TripleGeo, it does not include any support for reverse transformation, and its scalability against large geodatasets may suffer because of the complexity of RML mappings. Most importantly, GeoTriples also lacks any specific support for transformation of POI data (e.g., classification schemes). Besides, mappings generated by GeoTriple can be used by the *Ontop-spatial* extension [BK16],[BXK+16] of the Ontology-Based Data Access (OBDA) system *Ontop* [RR15]. This way, users can view their data sources virtually as linked data through on-the-fly GeoSPARQL-to-SQL translation on top of relational databases using ontologies and mappings. This is similar in spirit with Sparqlify, and this process does not require any transformation of data.

Towards developing technical specifications for the representation of *Points of Interest on the Web*, a working group was created, initially under the auspices of W3C, but later moved under the OGC. Although still work in progress [OGC-POI], this representation does not cover the issue of transforming POIs from external sources to RDF. In a separate approach, the SDI4Apps project [SDI4Apps] has also developed a data model for POI data and have created a SPARQL endpoint containing millions of POI features, but neither do they explain how this data have been mapped to their ontology nor provide any details on their transformation to RDF. Of course, there are tools that enable extraction of POI entities from datasets. For example, [OsmPoisPbf] scans an OpenStreetMap (OSM) file for nodes and areas (and relations) whose tags indicate them as POIs and extracts those into a text CSV file. To the best of our knowledge, no tool for direct transformation from POI representation to RDF is available.

As we elaborate in the sequel, our own transformation utility TripleGeo [TripleGeo] was initially based on Geometry2RDF [geo2rdf], but since its inception it has been substantially enhanced by integrating access to several external geospatial file formats and DBMSs and providing support for GeoSPARQL data types. In its current release (ver.1.4), TripleGeo includes support for attribute mappings according to a given ontology, assignment of categories according to a classification scheme, as well as customizations specifically targeting transformation of POIs.

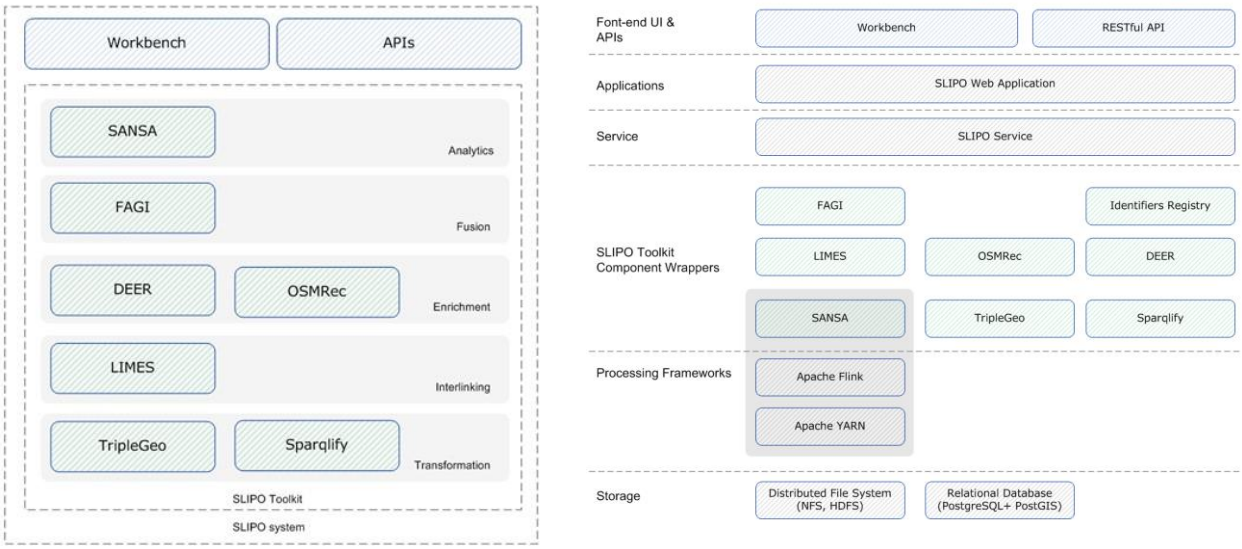
1.3. SLIPO Transformation Framework

The SLIPO POI integration lifecycle is realized through the SLIPO Workbench, a platform for defining, executing and managing POI integration workflows. These workflows integrate all the components of the SLIPO toolkit, supporting the integrated execution of all four core POI integration steps: transformation, interlinking, enrichment and fusion. Additionally, the SLIPO system prescribes a set of value added functionalities on top of integrated POI datasets (Figure 2 (a)). SLIPO proposes an innovative approach for addressing the POI data integration challenges. Modelling of POIs is based on flexible, enhanced and semantically rich RDF profiles, allowing encapsulation of diverse attributes and metadata from different sources, as well as to support the representation of POI evolution, provenance, and quality. This will be accompanied by transformation tools that will allow to import and export POI data in RDF, allowing to transfer and tackle the data integration challenges in the Linked Data domain, while still maintaining compatibility with existing formats and products. Once transformed into RDF, the POI data may be interlinked and fused to identify and resolve duplicates and other relations (e.g., containment). The SLIPO toolkit (shown in Figure 2(b)) will be a complete suite of integrated software and services for POI data integration, which will support all steps of the POI lifecycle and stages of the data integration process. Regarding transformation of conventional POI data, metadata, and schemata into RDF according to a comprehensive ontology, the SLIPO toolkit will offer two distinct utilities, namely *Sparqlify* and *TripleGeo*.

Sparqlify [SPRQLF] is a robust, scalable SPARQLSQL query rewriter that allows the definition of RDF views using a Sparqlification Mapping Language (SML). This way, it enables SPARQL queries on relational databases, serving as a SPARQL endpoint for billions of virtual triples from the underlying database. Sparqlify is the underlying framework that supports the LinkedGeoData (LGD) project. LGD serves OpenStreetMap as an RDF knowledge base according to the Linked Data principles. Further, it implements a mapping between the OSM categories hierarchy and its OWL ontology counterpart. Sparqlify will be used as the basis for

creating mappings between conventional POI schemas and the global POI ontology, using its SML language. Sparqlify will be utilized in scenarios where big datasets of POIs are stored in DBMSs, and require heavyweight processing in order to be mapped and transformed into RDF data. Currently, Sparqlify is using the SPARQL-to-SQL rewriter on top of Apache Spark within the SANSA platform [SANSA], which is employed at the *analytics stage* in SLIPO. Sparqlify will be integrated within SANSA, so that a SPARQL query to SANSA is undertaken by Sparqlify, and then this is being rewritten to SQL and executed on Spark. More details regarding Sparqlify and its use in SANSA will be available in Deliverable D4.1.

TripleGeo [TripleGeo] is an ETL utility that can extract geospatial features from various sources (e.g. shapefiles, spatial DBMSs) and transform them into GeoSPARQL-compliant RDF triples. It copes with most common spatial data types, like points, linestrings and multi-linestrings, polygons and multi-polygons and supports on-the-fly transformations between different coordinate reference systems. In SLIPO, we take advantage of TripleGeo's support of several geospatial representation and Coordinate Reference Systems to handle the heterogeneity of POI formats and representations. Further, we further enhance the modular architecture of the tool and we aim to extend it as a distributed processing solution in order to be able to handle massive amounts of POI datasets.



(a) High level architecture (b) Logical architecture

Figure 2: SLIPO Architecture

1.3.1. Achievements by M15 – TripleGeo ver.1.4

Version 1.4 is the currently available release of our ETL software TripleGeo. Based on the source code of its ver.1.1, which was originally developed in the context of the GeoKnow project [GeoKnow], there have been three releases in the framework of the SLIPO project with particular focus on POI-specific mapping and transformation functionalities. Next, we enumerate the new features, as well as the improved functionality and performance of TripleGeo, as a result of our work until M15 of the project.

1.3.1.1. New or Improved Functionality

- **Improved geospatial support**. Currently, TripleGeo supports extraction of data from various geospatial repositories, either de facto geographical file formats (e.g., ESRI shapefiles, GML, CSV,

etc.), or geospatially-enabled DBMSs (e.g., Oracle Spatial, PostGIS). This includes improved handling not only of primitive geometry types (like points, linestrings, or polygons), but even more complex geometries (MultiPolygons, Geometry Collections), as well as on-the-fly transformation to another coordinate reference system (CRS reprojection).

- **Compliance to standardization initiatives**. This mostly concerns RDF geometries, which can be fully *compliant with the OGC GeoSPARQL standard* (2012), and indirectly to other OGC standards [OGC07],[OGC10]. Moreover, TripleGeo provides support for *INSPIRE-aligned data/metadata* [INSPIRE], abiding by the INSPIRE Directive by the European Commission that sets the legal and technical foundations towards interoperable Spatial Data Infrastructures across Europe.
- **User-defined mappings for transformation of thematic attributes**. Such mappings from original attributes to properties according to a given ontology, allow transformation of all available attributes per input entity.
- **Customized URIs**. TripleGeo allows auto-generation of UUIDs for each transformed feature, so that these can be used for assignment of URIs according to namespaces prescribed by an underlying ontology (specified by the user).
- **Reverse transformation from RDF to de facto geographical file formats**. This utility enables users to obtain a certain amount of the semantic information and metadata as attributes in conventional geodata formats (like ESRI shapefiles or CSV).
- **Specialization on POI data**. Although TripleGeo was conceived and remains a *general-purpose* ETL tool to the RDF data model, we have included specific support for transformation of POI data into RDF. So, the current version includes support for RML mappings to handle extra thematic attributes into RDF according to the SLIPO POI ontology. For improved efficiency, we have also prepared an alternative mapping facility specifically tailored for SLIPO ontology. TripleGeo also provides support for hierarchical classification schemes regarding POI categories and also yields auxiliary output of basic attributes per POI feature in order to be used by the SLIPO Identifiers Registry.
- **Integration with the SLIPO Workbench**. TripleGeo has been successfully integrated into the SLIPO toolkit and POI data integration workflows. This allows users to interact with the tool in a coherent, user-friendly, and flexible manner under the entire SLIPO data lifecycle. Details on this integration will be provided in Deliverable D1.3.

1.3.1.2. Performance

- **Efficiency.** We have conducted a series of tests regarding both transformation and reverse transformation against a variety of geospatial repositories, confirming that TripleGeo can now handle any number of thematic attributes and map them to a given POI ontology. It should be also noted that we have prepared configurations and mappings specifically for transforming datasets supplied by our SLIPO partners, verifying that TripleGeo can offer consistent RDF representations for POI data from diverse providers.
- **Scalability.** Experimental results testify that TripleGeo ver.1.4 has already achieved orders of magnitude performance gains compared to its original release, and can now efficiently transform millions of POIs in a few minutes even without any sophisticated data partitioning schemes, thus paving the way for even more advanced scalability in its forthcoming releases.

2. The TripleGeo Software

TripleGeo [TripleGeo] aims to bridge the gap between typical geographic representations from a variety of proprietary files, DBMSs, and georeferenced systems with the demands of geospatially-enabled RDF stores. Although it was initially developed based on open-source *geometry2rdf* library [geo2rdf], notable modifications and substantial enhancements have been made in order to meet interoperability needs in RDF stores. In fact, TripleGeo is designed as a *spatial ETL tool*, enabling users to:

- *Extract* spatial data from a source;
- *Transform* this data into a triple format and geometry vocabulary prescribed by the target RDF store;
- *Load* resulting triples into the target RDF store.

Therefore, TripleGeo always preserves data integrity and provides consistent, well-defined geospatial information to end users. This tool can take as input not only de facto geographical files (e.g., 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 polygons, but also more complex geometries (e.g., geometry collections). 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 GeoSPARQL standard [OGC12]. Users can control the transformation process through a configuration file that offers ability to parameterize execution and define user-specified settings (e.g., namespaces, serialization format, georeferencing, etc.). Last, but not least, latest versions of the software enable extraction of all thematic (i.e., non-spatial) attributes available in the input data, as well as definition of classification schemes for assigning categories in the input entities.

In the context of SLIPO, several customizations and extensions have been applied in the TripleGeo source code and functionality in order to support transformation of POI data (e.g., support for mappings to the SLIPO ontology [SLIPO-D2.1]). Although these add-ons are focused on POI data transformation, it should be stressed that TripleGeo remains a general-purpose ETL tool that can be used for efficient transformation of any geospatial features in vector format.

In this Section, we first discuss the original ver.1.1 of TripleGeo by briefly outlining its components and processing flow as these were available at the beginning of the SLIPO project (M1). Next, we present our plans for its ver.2.0 that will become available at the end of the project (M36). We foresee that TripleGeo ver.2.0 will include advanced functionality and versatility in accessing a multitude of geospatial data formats, as well as orders of magnitude performance improvements in handling large POI datasets. Finally, we present an overview of the features already available in the current ver.1.4 of the software.

2.1. TripleGeo ver.1.1

As discussed in Section 1.2, several ETL tools have been available for converting between geospatial formats, but only a few specifically addressing the emerging needs of geospatially-enabled RDF stores. In 2013, in

the context of the GeoKnow project [GeoKnow], we began developing TripleGeo, an open-source ETL utility that can extract geospatial features from various sources and transform them into triples for subsequent loading into RDF stores. At that time, TripleGeo was the first tool that enabled conversion of geospatial features from several sources and formats into GeoSPARQL-compliant serializations according to the OGC GeoSPARQL standard [OGC12].

2.1.1. Legacy Implementation

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 in the latest version of *geometry2rdf* (dating back to 2012 and available at [geo2rdf]) offers 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 had provided a stable base for developing our initial ver.1.0 of TripleGeo in 2013, mostly geared towards integration of a few geospatial data repositories and support for GeoSPARQL types.

2.1.2. Basic Features and Functionality

More specifically, the aim of TripleGeo was to bridge the gap between typical geographic representations from a variety of proprietary files, DBMSs, and georeferenced systems with the demands of geospatially-enabled RDF stores.

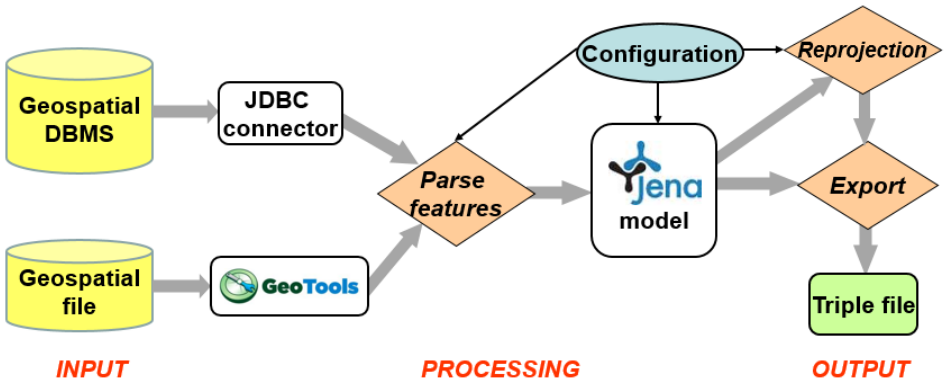


Figure 3: Processing flow diagram for transformation to RDF as applied with TripleGeo ver.1.1

Figure 3 illustrates the processing flow used for converting geospatial features into RDF triples with TripleGeo ver.1.1. Among its distinctive features, we point out that ver.1.1 can:

- Directly access de facto geographic formats (e.g., ESRI shapefiles, GML, KML) or DBMSs (IBM DB2, MySQL, Oracle Spatial, PostgreSQL/PostGIS).
- Recognize many geometric data types, i.e., not only points, but also (multi)linestrings and (multi)polygons.
- Extract a *limited* number of *thematic attributes* associated with each feature. In practice, TripleGeo ver.1.1 can handle up to four attributes per feature: its *geometry* (*mandatory*), a unique *identifier* (*mandatory*) for each entity and will be used for identifying the extracted resource, as well as two optional string values concerning its *name* and a *type* that characterizes this entity (e.g., "restaurant").
- Allow on-the-fly reprojection between any established Coordinate Reference Systems (CRS), e.g., transform geometries from GreekGrid87 to WGS84.
- Provides integrated transformation of INSPIRE-aligned spatial data and metadata into RDF using XSL stylesheets for selected INSPIRE data themes [INSPIRE]. This allows geospatial data (standards-compliant or not) to be transformed to RDF and exposed through GeoSPARQL with limited effort.
- Export triples into various serializations (RDF/XML, NTRIPLES, TTL, etc.) and geometry vocabularies for swift loading into RDF stores.

From a user's perspective, this command-line utility is entirely automated and based on preconfigured settings. A configuration file declares user preferences concerning all stages of the conversion: how input source will be accessed, which data is involved, what geometric representation should be used, whether geometries must be reprojected into another CRS, as well as the output triple notation.

Since its inception, TripleGeo is open source software and it can be redistributed and/or modified under the terms of the GNU General Public License [GPL].

2.1.3. Performance

Regarding ver.1.1 of TripleGeo, we had conducted a performance study in the context of the GeoKnow project [GeoKnow-D2.2]. Indicatively, we had noticed that transformation took less than 3 minutes for an OpenStreetMap layer with around 590,000 point geometries (including cost of writing the RDF triples to disk files). In contrast, 2,600,000 linestring geometries required much more time, about 2.5 hours to conclude the transformation. Such delays should be mostly attributed to memory shortage, as the entire RDF model in ver.1.1 was retained in main memory and grew proportionally to the amount of statements generated per initial record, so it had to spill on disk in case of excessive load. 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, as in many modern processing paradigms and exactly as we intend to do in the context of SLIPO.

2.2. Towards TripleGeo ver.2.0

Next, we present the planned additions to the functionality of TripleGeo, as inferred by the SLIPO requirements, and we discuss some specific issues and targeted actions that will guide the implementation of its next major release (ver.2.0) to be delivered at the end of the project as part of the SLIPO Toolkit.

Thanks to its modular implementation, TripleGeo can be enhanced with more utilities without affecting existing functionality. During the project, we have begun to further extend TripleGeo with several novel features, and most importantly, specific functionalities that can support the scalable transformation of large POI datasets. Next, we present the new features that will be implemented in ver.2.0, the usability options in the data integration lifecycle and performance improvements to satisfy our scalability goals. We also indicate a time schedule for their implementation in two stages: an *intermediate* functional release that is already available (M15) for integration with the SLIPO Service and Workbench, as well as the final full-fledged version (M36) that will cover all transformation requirements for SLIPO.

2.2.1. New Features

The new functionality to be provided by TripleGeo ver.2.0 will include:

- Ability to apply mappings and vocabularies and export both geometric and thematic attributes of the original dataset under a given OWL. This will enable incorporating and deploying mappings between identified concepts and properties of the source data and the target ontology into the transformation mechanism. For compatibility, such mappings should ideally be expressed in a widely used mapping language (like RML [RML], R2RML [R2RML], or D2RQ [D2RQ]) and we will apply best practices in the way original features are mapped into RDF concepts and properties. We will also examine whether TripleGeo should be extended with semi-automatic workflows for guiding the user into creating new mappings (e.g., in transforming datasets whose schema is not mapped to an existing POI ontology) or this functionality should preferably become an autonomous tool in the SLIPO architecture. [*Intermediate release*: M12, *Final release*: M36]
- Integration with URI identifier creation. The methodology that will be developed in SLIPO for creating persistent, unique, vendor and technology independent POI URIs will be integrated in the transformation process performed in TripleGeo. Furthermore, this functionality will become available for any other source dataset (i.e., even non-POI data) and replace the existing hard-coded creation of URIs by user-specified namespaces. This will allow users to customize the creation of linked identifiers and to choose a specific naming strategy for URIs (e.g., as a combination of metadata of the original feature). [*Intermediate* release: M12, *Final release*: M36]
- Interaction with other geographic data sources (e.g., GPX, CSV), de facto POI formats (like TomTom Overlay files [TToverlay], OziExplorer Waypoints [Ozi] etc.) and DBMS platforms (e.g., MS SQL Server spatial, SpatialLite). [*Intermediate* release: M12, *Final release*: M24]
- Support for more complex geometric types (e.g., geometry collections) is equally important for acquiring the most detailed representation of spatial entities instead of point coordinates of centroids. [*Final release*: M12]

The usability options of TripleGeo in the SLIPO POI data integration lifecycle involve:

- Specialization to POI transformation is of major importance for SLIPO, so TripleGeo will be extended and optimized to effectively support POIs, with specific vocabularies and operations. TripleGeo will be aligned with the global POI ontology to be developed in SLIPO, thus allowing representation of more complex POI metadata and relations to facilitate the POI integration lifecycle. This way, TripleGeo should be able to provide support for vocabularies and mappings specifically for POI data handled in SLIPO (e.g., TomTom). Furthermore, we will examine whether TripleGeo should also need to handle vocabularies and mappings to existing custom POI schemata (e.g., OSM, Wikimapia) for effective manipulation of large-scale open geodatasets. [*Intermediate release*: M12, *Final release*: M36]
- Integration with the SLIPO Workbench, will provide users with a GUI or via a RESTful API that facilitates customization of the transformation process. Instead of the current command-line interface, this GUI may expose the full functionality of TripleGeo for disk-based files, tables in a DBMS or web-accessible data, and offer a unified web interface to extract large POI datasets, convert them and produce their RDF representations. TripleGeo can be thus incorporated into the SLIPO toolkit and POI data integration workflows in a coherent, user-friendly, and flexible manner under the entire SLIPO data lifecycle (i.e. transformation, interlinking, fusion, and enrichment of POIs). [*Intermediate release*: M12, *Final release*: M36]
- Last, but not least, reverse transformation from RDF into de facto geospatial formats is required. TripleGeo will support reverse transformation of RDF POI data (potentially interlinked or fused in later stages) into de facto POI formats (e.g., shapefiles, GPX, CSV, etc.). Of course, there exists an impedance mismatch in this direction, given that the POI ontology is semantically more expressive than the conventional POI schemata, thus POI attributes, relations and metadata will be richer than what can be supported by conventional formats. To address this issue, we will define and implement the optimal reverse transformations that will allow the incorporation of the maximum amount of semantic (linked, enriched, fused) POI information and metadata into the available properties of conventional POI formats. [*Intermediate release*: M12, *Final release*: M24]

2.2.2. Scalability

Scalability with increasing data volumes is most challenging. Hence, a parallelization framework in transforming input features and generating RDF triples would be advantageous. Assuming that *n* processing nodes are available, we will opt for solutions that employ data partitioning of the input into *n* disjoint batches, so that TripleGeo can be invoked in *n* separate instantiations, each one executed in a processing node in isolation from the rest. Such a partitioning may be based on several alternative schemes. For example, the original file may be split into *n* smaller batches with equal number of records in each one. Besides, splitting be based on spatial criteria, e.g., employing a subdivision of the space into *n* disjoint regions (e.g., a tessellation into cells), and creating a batch for each region with all entities contained therein. Of course, each original spatial entity with all its attributes will be included in only one such subset. Such partitioning has absolutely no impact on transformation to RDF, as each entity is transformed independently from the rest. By employing *n* concurrently running instantiations of TripleGeo to transform each dataset, and emitting triples into flexible RDF storage schemes (e.g., HDFS), we expect that scalability of transformation will be greatly improved.

According to our planned framework, our goal at the end of the project is to support RDF transformation of *100 million POIs in less than a minute*, thus aiming to achieve orders of magnitude performance gains compared to the edition of the software available at the beginning of the project. [*Final release*: M36]

2.3. Current Features and Functionality

Table 1 outlines all major features and functionality available in the successive releases of TripleGeo since the beginning of the project. Note that ver.1.1 is practically the software as inherited from the GeoKnow project, whereas all subsequent releases have been prepared in the context of SLIPO with the principle objective to offer support *specifically for transformation of POIs into RDF*. However, note that TripleGeo remains a *general-purpose ETL tool*, and its support for transformation into RDF is not limited to POI datasets only, but may be useful to many other application domains (transport networks, administrative areas, hydrography, cadastre, etc.).

	Functionality	ver.1.1 (M1)	ver.1.2 (M7)	ver.1.3 (M11)	ver.1.4 (M15)
GENERAL-PURPOSE	Access to de facto geographical file formats	4	6	8	8
	Access to geospatially-enabled DBMSs	4	5	6	8
	INSPIRE-aligned data/metadata	YES	YES	YES	YES
	Process complex geometries	NO	YES	YES	YES
	CRS reprojection	YES	YES	YES	YES
	Compliance to OGC GeoSPARQL standard	YES	YES	YES	YES
	Export to various RDF serializations	YES	YES	YES	YES
	Extraction of thematic attributes	MINIMAL	MINIMAL	ALL	ALL
	Auto-generation of UUIDs & assignment of URIs	NO	NO	NO	YES
	Reverse transformation to de facto geographical file formats	NO	NO	NO	YES
POI-SPECIFIC	POI Classification schemes	NO	NO	MINIMAL	YES
	RML mappings for the SLIPO ontology	NO	NO	YES	YES
	Custom YAML mappings for the SLIPO ontology	NO	NO	NO	YES
	Provide updates to the SLIPO Registry	NO	NO	NO	YES

Table 1: Functionality supported by successive versions of TripleGeo

General-purpose features. In particular, TripleGeo has evolved considerably until M15 of the project and currently supports a wide range of *general-purpose* functionality:

- Access to various *geospatial repositories* for extracting data:
    - Compared to the originally supported four *de facto geographical file formats* (ESRI shapefiles, GML, KML, XML in ver.1.1), more have been added in successive releases: CSV, GeoJSON (in ver.1.2), GPX and OSM XML (in ver.1.3).

    - Four *geospatially-enabled DBMSs* were supported in ver.1.1 (namely, IBM DB2, MySQL, Oracle Spatial, PostgreSQL/PostGIS), but more have been added in successive releases: Microsoft SQL Server (in ver.1.2), SpatiaLite (in ver.1.3), as well as ESRI personal geodatabases and Microsoft Access format (in ver.1.4).

- Improved handling of *geometries*:
    - Since its inception, TripleGeo aimed to support *various geometry types*, and not only points. Hence, even in ver.1.1 it can process representations of all primitive types for 2-dimensional geometries (Point, MultiPoint, LineString, MultiLineString, Polygon, MultiPolygon), but even more complex geometries (Geometry Collection) are supported since ver.1.2.
    - On-the-fly transformation of the coordinate reference systems (*CRS reprojection*) has been one of the distinctive features of TripleGeo, as it allows consistent representation and search over geospatial information that may be possibly collected from various sources using differing georeferences.
- Compliance to *standardization* initiatives:
    - Upon its initial release (ver.1.0, June 2013), TripleGeo was the first tool that allowed transformation of geospatial data into RDF geometries fully *compliant with the OGC GeoSPARQL standard* (2012). Of course, such compliance also entails support for geospatial representations according to OGC standards [OGC07], [OGC10], and it is certainly preserved in all subsequent releases of the software.
    - Support for *INSPIRE-aligned data/metadata*: Since ver.1.1, TripleGeo supports extraction of INSPIRE-aligned GML data for several thematic domains, as well as XML metadata and can transform them into RDF triples with geometries. The current ver.1.4 includes minor improvements in the usability of this feature.
- Exporting transformed data into a variety of *RDF serializations* is also supported. However, note that some transformation modes impose restrictions in the serialization format for performance reasons, as will be discussed in Section 3.3.4.
- Until ver.1.3, TripleGeo had minimal support for extraction of *thematic attributes*. In fact, at most three attributes could be extracted for a given spatial entity, and these were restricted to a unique identifier, its name, and its category, all represented as string literals with standard RDF properties (like `rdfs:label`, `rdfs:type`) and not allowing any customizations with an OWL ontology. Since ver.1.3, TripleGeo supports *user-defined mappings* from original attributes to properties according to a given ontology, thus allowing transformation of all available attributes per input entity. Although we have specific provision for mappings tailored to the SLIPO ontology (explained next), we stress that such mappings are not limited to POIs, but may be defined for other geospatial data as well (e.g., road networks, administrative areas) and then applied for their transformation to RDF.
- Until ver.1.3, URIs of all entities transformed with TripleGeo were based on original identifiers in the input data; if not present, then there was no guarantee that different entities would not end up having the same URIs. Starting from ver.1.4, and complying with the requirements of SLIPO, TripleGeo auto-generates *UUIDs* per input entity and accordingly assigns a URI using namespaces

prescribed by an underlying ontology (specified by the user). Thus, it accomplishes to assign stable, well-formatted, globally unique, persistent, and manageable URIs to the resulting triples.

- Starting from ver.1.4, TripleGeo offers the ability for *reverse transformation* from RDF to de facto geographical file formats. This utility enables users to obtain a certain amount of the semantic information and metadata as attributes in conventional geodata formats (like ESRI shapefiles or CSV). Although this is a special requirement within the context of SLIPO for is specific to the POI integration lifecycle, so that semantic data (possibly interlinked, fused, and enriched) can be accessible and exploitable by existing GIS software and services, TripleGeo supports reverse transformation not only for POI datasets, but also for other types of geospatial linked data queryable with SPARQL according to a consistent ontology.

POI-specific features. Besides, TripleGeo also offers support for the following features *specifically regarding POIs*:

- Typically, data providers employ diverse *classification* or *tagging schemes* to categorize POIs and describe their type, as this is critical for applications based on POI data (e.g., searching for restaurants). TripleGeo ver.1.4 accepts specification of (possibly hierarchical) classification schemes for POIs, produces RDF triples that fully describe this information along with especially assigned URIs, and introduces extra links between a POI and its respective category under this scheme.
- Regarding extraction and transformation of *thematic attributes* of POIs, TripleGeo currently supports *mappings to the SLIPO ontology* in two alternative formats:
    - *RML mappings* are expressed in the RDF Mapping Language [RML] and specify rules that dictate how to generate RDF triples from input data. This is available since ver.1.3 and can be applied when TripleGeo is executed in the *RML transformation mode* (Section 3.3.4.3).
    - *YAML mappings* are less expressive than the aforementioned RML mappings, but they are simpler to specify and also customized for the SLIPO ontology. Most importantly, thanks to their simplicity, they allow very fast transformation of scalable volumes of POI datasets. This king of mappings is available with ver.1.4 and is applicable when TripleGeo is executed either in *GRAPH* (Section 3.3.4.1) or *STREAM transformation mode* (Section 3.3.4.2).
- The *SLIPO Registry* is a directory of identifiers created and associated with key metadata about each POI processed within the POI integration lifecycle. To support this, TripleGeo compiles the requested properties per POI during transformation (including its URI, name, category, and location) and eventually creates a file that can be acquired and processed by the SLIPO Registry in order to update this directory.

2.3.1. Libraries and Frameworks

Since its inception, TripleGeo includes dependencies to various open-source tools and libraries, all of which are used "as is". The most significant of these libraries and frameworks 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. 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;
- stores to allow large numbers of RDF triples to be efficiently stored on disk;
- 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.
- ***RDF Mapping Language (RML).*** This is a generic mapping language [RML] defined to express customized mapping rules from heterogeneous data structures and serializations to RDF. It is defined as a superset of R2RML, the W3C recommendation for a mapping language from databases [R2RML], and aims at extending its applicability to a broader variety of input sources (principally, CSV, XML, and JSON formats). RML provides a generic way to define mappings easily transferable to cover references to other data structures, combinable with case-specific extensions, and always backward compatible with R2RML. RML is open source and is released under the MIT license.

2.3.2. Licensing

The TripleGeo tool is open source 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. TripleGeo 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].

2.3.3. Documentation

A JavaDoc with full API documentation has been prepared in HTML format regarding the Java source code of TripleGeo ver.1.4 and it is publicly available [TripleGeo]. This documentation covers all classes implemented for TripleGeo with specific details for all their methods and data structures.

3. Transformation to RDF

Transformation of conventional POI and third-party datasets into RDF is an essential part of the POI data integration lifecycle, as it enables their subsequent processing (interlinking, fusion, enrichment) as linked data. Towards this goal, TripleGeo offers advanced capabilities for transformation from a large variety of data sources that include world-renowned DBMSs and geospatial de facto file formats.

In this Section, we provide a detailed account of the processing flow for transforming POI data to RDF as applied by TripleGeo ver.1.4. We also present its core components, including the implemented functionality that supports attribute mappings and classification schemes, as well as the various transformation modes. Further, we outline the complementary mechanism for registering POIs in the SLIPO Identifiers Registry based on output prepared by TripleGeo. Finally, we explain our initial scheme for achieving scalability over large geospatial datasets with multiple worker threads of TripleGeo each processing disjoint subsets of the input data.

3.1. Architecture

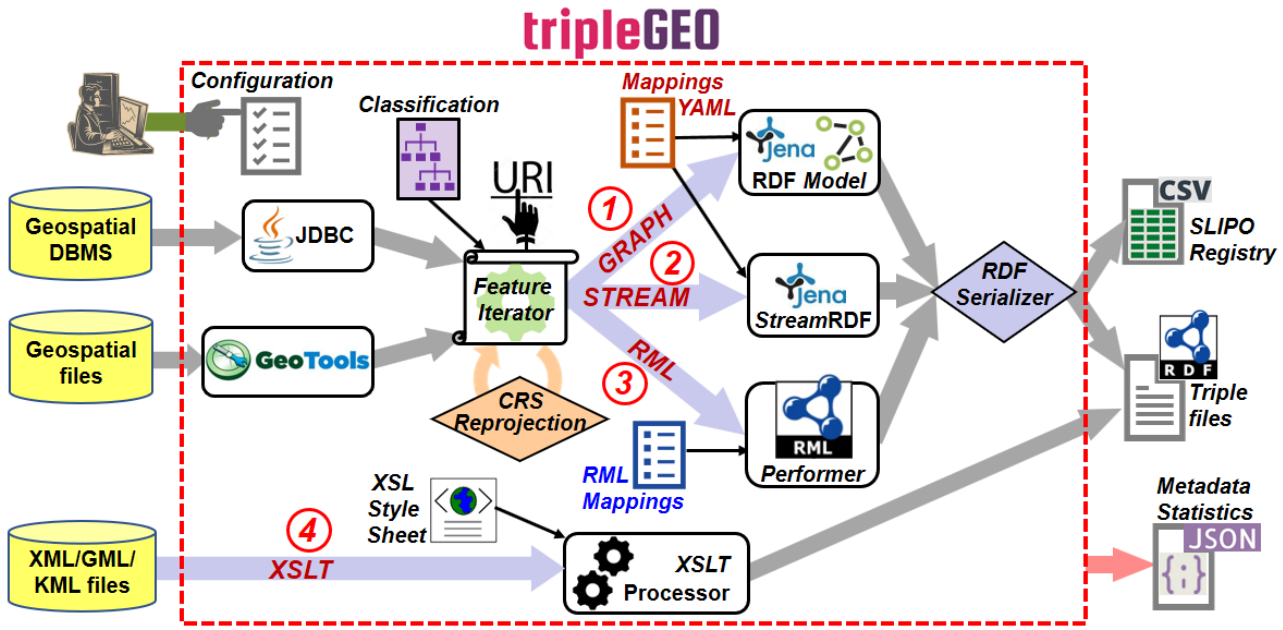


Figure 4: Processing flow for transformation to RDF with TripleGeo ver.1.4

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 a straightforward fashion according to some preconfigured settings. Figure 4 illustrates the flow diagram used for converting geospatial features into RDF triples. Next, we outline the basic components of the utility:

Input geospatial data may be obtained from geospatial files either *structured* (e.g., shapefiles, CSV) or *semi-structured* (in XML, GML, or KML), as well as directly from geospatially-enabled DBMSs. Currently, TripleGeo can access features stored in seven DBMS (Oracle Spatial, PostGIS, etc.).

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 all properties that control the various stages of transformation: 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 5.2.1.

As detailed later in Section 3.3.4, transformation to RDF with TripleGeo ver.1.4 can be carried out in four different *modes*:

1. *GRAPH*, which makes use of a disk-based Jena model to store all transformed triples;
2. *STREAM*, which applies in-memory conversion with prompt creation of triples per input feature;
3. *RML* for applying on each input feature custom attribute mappings specified by the user in the RDF Mapping language [RML]; or
4. *XSLT* specifically for parsing semi-structured data according to user-provided *XSL style sheets* and generating RDF triples.

Especially for *structured* data obtained either from files or retrieved from a DBMS, a *feature iterator* consumes each input record (i.e., all attributes concerning a POI) and converts its geometry into a suitable representation according to user specifications. Optionally, *reprojection* of geometries into another spatial reference system (CRS) is available. This coordinate transformation is carried out thanks to the integrated GeoTools library and according to user specifications for the source and target CRS.

Regarding *thematic* (i.e., non-spatial) attribute values (e.g., type, name, contact information) of an input feature, TripleGeo emits properly formatted literals as defined in user-specified *attribute mappings*. Depending on the transformation mode, these mappings can be prescribed in two alternative representations, either using RML or in a custom YAML format. In either case, such mappings should reflect the underlying ontology (i.e., the SLIPO POI ontology [SLIPO-D2.1]).

TripleGeo assigns a *URI* to each processed feature according to user specified scheme in the configuration. Namespaces for classes can be also defined in the configuration and used in the transformation according to the mappings. TripleGeo intentionally avoids creating blank nodes by inheriting the URI of the main feature to all its object properties with a suitable suffix.

In case that input data follows a (possibly multi-tier) *classification scheme* into categories, subcategories, etc. assigned to features, this can be also utilized in the transformation by assigning URIs to these categories. The classification scheme is also transformed into triples (according to the hierarchical structure defined in the SLIPO ontology) and each transformed entity (i.e., POI) is linked to its respective category.

Serialization of generated triples into export 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. In addition, an extra CSV file is

issued, containing basic attributes per input feature (e.g., URI, name, category, geometry of a POI) specifically for registering it in the *SLIPO Registry* [SLIPO-D2.1].

Finally, *metadata statistics* compiled during the entire transformation process are written into another JSON file. This metadata concern performance, input and output size, number of transformed values per attribute, and the spatial extent of the data.

3.2. Input and Output

3.2.1. Input

TripleGeo has been implemented for accessing geospatial data from a wide range of file or DBMS repositories. As of M15, its current release ver.1.4 has been successfully tested in both MS Windows and Linux environments and can provably deal with *structured or semi-structured geospatial data* stored in one of the following *vector file formats*:

- ESRI *shapefiles* [ESRIshp], one of the most popular geographical data formats supporting primitive geometric data types for (Multi)Points, (Multi)LineStrings, and (Multi)Polygons, along with their thematic attributes;
- *GeoJSON* files in JavaScript Object Notation with geometry features including (Multi)Point, (Multi)LineString, and (Multi)Polygon [GeoJSON];
- *GPS Exchange Format* (GPX) that describe waypoints and tracks captured and stored by GPS devices and software [GPX];
- *CSV* containing geometries either as pairs of (x,y) coordinates (for points) or in WKT serialization (for any geometry type);
- OpenStreetMap XML files [OSM];
- *GML* files in Geography Markup Language [GML];
- *KML* files in Keyhole Markup Language [KML];
- *INSPIRE-aligned data* (in GML) and *metadata* (in XML) [INSPIRE].

Since version 1.2, TripleGeo is capable of consuming data from *multiple* geographical files and transforming them into RDF. The only precondition is that all input files must be in the same format (e.g., shapefile), with records in each file having the same attribute schema. TripleGeo handles separately each file and writes its resulting triples into a distinct file according to the user-specified serialization.

TripleGeo 1.4 is also able to retrieve data stored in the following *geospatially-aware DBMS* platforms:

- *Oracle* 12c Spatial and Graph [Oracle];
- *PostgreSQL* 9.4 with *PostGIS* 2.x [PostGIS];
- *MySQL* 5.6 [mySQL];
- IBM *DB2* 9.5 with Spatial Extender [IBM-DB2];
- SpatiaLite 4.3.0a [SpatiaLite];

- Microsoft *SQL Server* 2016 [MSSQLServer];
- ESRI *Personal Geodatabases* [ESRIGeoDB] in MS Access (.mdb) format; and
- Microsoft Access 2016 [MSAccess].

Geometric data must reside in a *single* table or view in the DBMS. Currently, there is no support for combining information from several sources (e.g., by joining two or more tables). Tests against most of these DBMSs have been conducted in both MS Windows and Linux environments; the last three DBMSs have been tested with TripleGeo in MS Windows only.

3.2.2. RDF Output

The main output of TripleGeo is the RDF triples returned after the transformation of the input 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 RDF formats:

- *RDF/XML*. This is the *default* output serialization 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. This is the default serialization available with either *STREAM* or *RML* modes, as this syntax does not use any internal state, it is the fastest to write in streaming fashion, and data of any size can be output. It also maximises the interoperability with other systems and are useful for database dumps. 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 legacy vocabularies for Virtuoso [VirtGeoRDF] and WGS84 RDF Geoposition vocabulary [GeoPos84], but note that this syntax is not compliant to GeoSPARQL.

3.2.3. Output to the SLIPO Identifiers Registry

In addition to RDF output, TripleGeo can optionally provide a CSV file having one record for each input feature (i.e., POI) with its basic attributes in order to be registered in the SLIPO Identifiers Registry as discussed in Section 3.3.5. Each such record consists of the following attributes:

- *URI* assigned to the POI;
- *Name* of the POI;
- *Category* according to a classification scheme accompanying the original data;
- *Data source* provider of this POI (e.g., OpenStreetMap);
- *Unique identifier* of this POI at the original data;
- *Location* specified as a pair of longitude/latitude coordinates in WGS84.

3.2.4. Metadata Statistics

Upon termination of a transformation process, TripleGeo provides *metadata* regarding its execution in a JSON file. As depicted in Figure 5, this file reports performance measurements and quality indications regarding the data listed in three main categories:

- *Attribute Statistics*: For each attribute in the original input dataset, a count of *NOT NULL values* on this attribute is given. This reflects the amount of such values that have been actually given to transformation, but note that the number of resulting triples may be inflated depending on the specified mappings.
- *Spatial Extent*: A Minimum Bounding Rectangle (*MBR*) as computed by the spatial extent that covers all transformed geometries. Note that this rectangle is always reported in WGS84 coordinates, irrespective of the spatial reference system (CRS) of input and output datasets.
- *Execution metadata* include the following items:
    - Count of output triples;
    - Count of input records (for structured data) or features (for semi-structured data);
    - End-to-end execution time (in milliseconds) that includes the cost of accessing and fetching the input data, its transformation cost, as well as the cost of writing output to file(s);
    - Path to the output file containing the resulting triples;
    - The RDF serialization of the output triples; and
    - The transformation mode employed in the execution.

This metadata may be utilized in the SLIPO Workbench in order to provide statistics and visualizations (maps, charts, etc.) about a given POI dataset.

```
Attribute Statistics:
    NAME_ENG:                          72355
    PHONE:                             72373
    ADD_NUMB:                          24034
    DATA_SOURCE:                       72373
    EMAIL:                             9400
    OBJECTID:                          72373
    NAME_GR:                           72373
    CATEGORY_URI:                      72373
    TK:                                42826
    ADDR_GR:                           45553
    FAX:                               12617
    ADDR_ENG:                          21205
    WEBSITE:                           17660
MBR of transformed geometries (WGS84):
    X_max:                             29.596572257377158
    Y_max:                             41.746653712939526
    X_min:                             19.374669876151085
    Y_min:                             34.80414849796334
Execution Metadata:
    Output triple count:               3313133
    Input record count:                72373
    Execution time (ms):               14240
    Output file:                       "./test/output/get-pois_v07.nt"
    Output serialization:              "N-TRIPLES"
    Transformation mode:               "STREAM"
```

Figure 5: Metadata resulting after transformation to RDF of a POI dataset with TripleGeo ver.1.4

3.3. Core Modules

In this Section, we provide important details concerning the design and implementation of core modules of TripleGeo regarding transformation of geospatial features to RDF.

3.3.1. Assignment of URIs

As specified in [SLIPO-D2.1], in SLIPO we opt to use HTTP URIs as POI identifiers, so that data owners have enough flexibility and full control over creating and managing their own POI identifiers, while still adhering to a uniform format. More specifically, TripleGeo ver.1.4 integrates support for constructing such URI identifiers according to recommended best practices. The URI pattern adopted is

```
http://{domain}/{type}/{concept}/{reference}[/{attribute}]
```

and consists of the following components:

- *domain*: As base for the URI in SLIPO, we prescribe `slipo.eu` as the domain.
- *type*: For identifying POIs, value `id` is used to identify the type of features.

- *concept*: In SLIPO, this may take several possible values (`poi`, `poiset`, `poisource`, `classification`, `term`) depending on the respective concept in the ontology where the identified resource adheres to.
- *reference*: This part is a 128-bit long number representing a *Universally Unique Identifier* (UUID) that is generated on-the-fly during transformation. We avoid reusing original POI identifiers or assigning auto-incremented integer values, since UUIDs can almost safely serve as unique identifiers without the need of a central registration authority or other means of coordination among involved parties.
- *attribute*: This is an *optional* component, used especially for generating *named URIs* regarding RDF properties of a POI feature, e.g., its name, address, category, etc. This suffix to the URI assigned to a POI is deliberately applied in order to avoid blank nodes in the resulting triples.

Note that TripleGeo does not check for duplicate POI identifiers either in the input data or in the assigned URIs. Naturally, in the context of POI data integration, it is possible that multiple representations of the same POI may be found in different sources. So, having to resolve duplicates during transformation when importing POIs in the system would not be practical and would have a significant impact on scalability. Instead, we intentionally opted to separate POI transformation from POI deduplication, allowing multiple identifiers for the same POI to be created and used, which may be interlinked at a later stage.

3.3.2. Attribute Mappings

As already mentioned, TripleGeo ver.1.4 offers two different options for specifying mappings from input attributes to RDF, i.e., either *RML mappings* (applicable in RML transformation mode) or custom *YAML mappings* (applicable in GRAPH/STREAM transformation modes).

Note that GRAPH/STREAM modes can also proceed to transform input data even if no YAML mapping has been specified. In this case, the name of an attribute in the input schema is turned into an RDF property (i.e., a predicate) in the ontology namespace defined by the user in the configuration. Certainly, such a flat mapping does not abide by a concrete ontology, but nonetheless it provides a valid RDF output that contains links between a central resource (the URI assigned to a POI) with all its known properties.

3.3.2.1. RML Mappings

In contrast to R2RML [R2RML] mapping language that can only define mappings of data from relational databases to RDF, the *RDF Mapping Language* (RML) [RML] offers extensions that support mappings also from structured or semi-structured file formats, such as CSV, XML, and JSON. RML is more expressive than R2RML and allows specification of mappings covering joins between input sources (e.g., master/detail tables). In contrast to R2RML, RML provides a vocabulary for defining a generic data source and the iterator pattern that specifies how the source data will be accessed. Such mapping definitions may be reusable across different sources (perhaps after small editing), as well as across different implementations for different source formats. In addition, RML allows cross-references among data from different input sources to be defined already on mapping level, by uniquely defining the pattern that generates a resource and refer to this definition any other time this resource is mapped.

More specifically, an RML mapping specifies rules concerning how input data will be represented in RDF as triples of (*Subject*, *Predicate*, *Object*) statements. This is achieved with a *TriplesMap* construct, which consists of three main parts:

1. The *Logical Source* covers many kinds of input data sources and specifies all necessary properties for accessing a data source and iterating over it.
2. A *Subject Map* defines how URIs will be generated for the specified resource. Such URIs will serve as subjects of all triples transformed according to this TriplesMap.
3. One or more *Predicate-Object Maps* can be defined for a given Subject Map. Each one consists of *Predicate Maps* that control how the respective predicate of triples is composed, as well as *Object Maps* or *Referencing Object Maps* that guide generation of objects in those triples

Note that all Subject, Predicate, or Object Maps are actually *Term Maps*, which control generation of RDF nodes, i.e., URIs or blank nodes or literals. A Term Map can be a *constant-value term map* that always generates the same RDF node, or *a reference-valued term map* that refers to a specific value (e.g., an attribute value) in the original data, or *a template-valued term map* which serves as a blueprint for transforming referenced columns from the source data. Furthermore, RML supports cross-references between Triples Maps, when the subject of a Triples Map is the same as the object generated by a Predicate-Object Map.

```
<#POIMapping>
    rml:logicalSource [
        rml:source "" ;
        rml:referenceFormulation ql:CSV
    ];
    rr:subjectMap [
        rr:template "http://slipo.eu/id/poi/{UUID}";
        rr:class slipo:POI;
        rr:class geo:Feature
    ];
    rr:predicateObjectMap [
        rr:predicate slipo:name;
        rr:objectMap [
            rr:parentTriplesMap <#POIName>
            ]
    ];
    rr:predicateObjectMap [
        rr:predicate slipo:lastUpdated;
        rr:objectMap [
          rml:reference "TIMESTAMP";
          rr:datatype xsd:dateTime
          ]
    ];
```

Figure 6: Fragment of RML mapping for POIs as used in TripleGeo ver.1.4

An indicative RML mapping for POIs in TURTLE format is given in the Annex (Section 9.3.1). A small excerpt of this RML mapping (`#POIMapping`) is depicted in Figure 6. Since TripleGeo utilizes its own configuration for accessing input data sources, we have simplified the specification of logical sources in RML mappings. So, the actual source (e.g., path to a CSV file) is omitted, whereas of RML's predefined Reference Formulations only the `ql:CSV` is declared. This latter specification actually stands as a proxy for column-oriented data formats, and does not mean that only CSV input is supported. Besides, *no iterator* needs to be specified, as TripleGeo provides one input feature (i.e., a POI) at a time and applies any transformations

specified in the RML mapping against its attribute values. In this example, a *Subject Map* is defined in order to generate the URI of each POI according to a *template-valued term map* that concatenates the SLIPO namespace with an attribute value (`UUID`) available in the input feature. Furthermore, extra triples will be generated, which associate this URI with specific nodes, i.e., that it is actually classified as a POI and also as a GeoSPARQL feature. Furthermore, a *Predicate-Object Map* is defined for the name of the respective POI, that will be generated via a separate *ParentTriples Map* specification (`#POIName`) and linked with the URI of that POI using a specific predicate (`slipo:name`). This is an example of *nested mappings*, where one mapping (`#POIMapping`) triggers another (`#POIName`). Another Predicate-Object Map for the timestamp of a POI is defined with a *reference-valued term map* that refers to a specific attribute (`TIMESTAMP`) available in the input record.

3.3.2.2. YAML Mappings

In the current release (ver.1.4) of TripleGeo, the aforementioned RML mappings are tightly coupled with the RML transformation mode and cannot be applied in any other mode. So, in order to support transformation of an arbitrary number of thematic attributes, we have introduced a *custom mapping* in YAML format to be used in GRAPH and STREAM modes. Our objective was to offer the minimal functionality required for mapping thematic attributes in an input dataset to the properties of SLIPO ontology [SLIPO-D2.1]. Of course, such a mapping may lack the expressiveness and generality of RML mappings, but it is simple and can be easily utilized by users in conversion to RDF under ontologies with moderate complexity like the SLIPO ontology for POIs.

An indicative YAML mapping for POIs is given in the Annex (Section 9.3.2). A small excerpt of this mapping is depicted in Figure 7. In practice, for each thematic attribute in an input feature that will be transformed into a resource, this YAML specifies up to seven properties:

- *entity*: This string value is used in creating a new RDF node for the given attribute value. This node inherits the URI of the POI it belongs to, suitably suffixed with this string value. As already mentioned in Section 3.3.1, this is a deliberate decision, as it avoids creation of *blank nodes* in the resulting triples, and also provides meaningful (and human readable) URIs for child nodes of POIs.
- *instanceOf*: This is used to specify RDF properties that are instances of classes. For instance, a name value should be transformed into an instance of the *Name* class in the SLIPO ontology, a phone value to an instance of the *Contact* class in the SLIPO ontology, etc.
- *partOf*: Specifies the name of *parent* (composite) RDF property that this resource is part of. For example, in Figure 7, the `osm_id` identifier will be transformed into a node that is part of the `sourceInfo` property of a POI according to the SLIPO ontology. Similarly, a street name is part of the address property, etc.
- *predicate*: This item specifies the namespace and the RDF property that will be used to link the URI of the POI feature with the RDF node that will be generated for this attribute.
- *type*: This is a characterization assigned as an extra (specialization) RDF property to a given node. For example, this may be used to specify that the name of a POI is "*official*", "*brand name*", "*international*", etc., or that a phone number is "*direct*", "*mobile*", etc.

- *datatype*: An XSD datatype that prescribes the correct interpretation of a string literal in the object of an RDF statement.
- *language*: Specifies the language tag (e.g., "*en*") to be used in string literals for this resource.

In transformation under GRAPH or STREAM modes, TripleGeo identifies the YAML mapping defined for a given attribute and accordingly creates RDF triples.

```yaml
osm_id:
  partOf: sourceInfo
  entity: source
  predicate: slipo:poiRef
name:
  instanceOf: name
  entity: name
  predicate: slipo:name
  type: official
  language: en
timestamp_:
  entity: lastUpdated
  predicate: slipo:lastUpdated
  datatype: timestamp
```

Figure 7: Fragment of YAML mapping for POIs as used in TripleGeo ver.1.4

3.3.3. Classification Schemes

Typically, POIs available by most data providers are classified to *categories* (e.g., restaurants, bars, theatres, etc.). This classification possibly follows a *hierarchical* scheme in multiple tiers, where each major category (e.g., FOOD) may be specialized into several subcategories (e.g., restaurant, fast food, pizza, etc.), etc.

Currently, TripleGeo supports three alternative ways to define a hierarchical classification scheme for POIs:

i. A user-prepared *CSV* file to be employed in assigning categories, subcategories, etc. to each feature in the input dataset. Attributes in this CSV file must be delimited with comma (',') and string values must be enclosed with double-quotes (e.g., `"RESTAURANT"`). Every item in the classification scheme must be defined with a unique identifier and name (e.g., `29,"RESTAURANT"`). Each line of the CSV file specifies a full path from a top-tier category to a bottom-level subcategory. E.g.: `5,"FOOD",29,"RESTAURANT"`. Figure 8 (left) depicts a fragment of a CSV-formatted classification scheme customized for OSM data, which is available in the Annex (Section 9.4.1).

ii. A user-prepared *YAML* file with indentations used to denote breakdown of a given category into subcategories (i.e., two blank characters in the beginning of a line at each extra level in the hierarchy). The identifier of each category (at any level) is specified after its name and it is preceded with a '#' character. Figure 8 (right) depicts a fragment of such a YAML-formatted classification scheme customized for OSM data, which exactly corresponds to the CSV fragment also listed in the same Figure and available in the Annex in full (Section 9.4.2).

iii. Especially for OpenStreetMap input data, a *YAML* file can be used to specify classification of features into categories according to their respective OSM tags. This classification may be employed when converting OSM XML data in either GRAPH or STREAM transformation modes. In TripleGeo, we currently make use of the OSM tags listed in [OsmPoisPbf] in order to classify OSM elements into custom categories.

```
"category_id","category","subcategory_id","subcategory"
1,"ACCOMMODATION",3,"CARAVAN"
1,"ACCOMMODATION",5,"HOSTEL"
1,"ACCOMMODATION",6,"HOTEL"
1,"ACCOMMODATION",7,"MOTEL"
1,"ACCOMMODATION",1,"ALPINEHUT"
1,"ACCOMMODATION",2,"CAMPING"
1,"ACCOMMODATION",4,"CHALET"
2,"AMENITY",8,"COURT"
2,"AMENITY",9,"FIRESTATION"
2,"AMENITY",11,"LIBRARY"
2,"AMENITY",12,"PLAYGROUND"
2,"AMENITY",13,"POLICE"
2,"AMENITY",14,"POSTOFFICE"
2,"AMENITY",15,"PRISON"
2,"AMENITY",16,"PUBLICBUILDING"
2,"AMENITY",17,"TOWNHALL"
4,"EDUCATION",19,"COLLEGE"
4,"EDUCATION",20,"NURSERY"
4,"EDUCATION",21,"SCHOOL"
4,"EDUCATION",22,"UNIVERSITY"
5,"FOOD",23,"BAR"
5,"FOOD",24,"BIERGARTEN"
5,"FOOD",25,"CAFE"
5,"FOOD",26,"FASTFOOD"
5,"FOOD",27,"ICECREAM"
5,"FOOD",28,"PUB"
5,"FOOD",29,"RESTAURANT"
```

```
ACCOMMODATION #1
  HOSTEL #5
  HOTEL #6
  MOTEL #7
  ALPINEHUT #1
  CAMPING #2
  CHALET #4
AMENITY #2
  COURT #8
  FIRESTATION #9
  LIBRARY #11
  PLAYGROUND #12
  POLICE #13
  POSTOFFICE #14
  PRISON #15
  PUBLICBUILDING #16
  TOWNHALL #17
EDUCATION #4
  COLLEGE #19
  NURSERY #20
  SCHOOL #21
  UNIVERSITY #22
FOOD #5
  BAR #23
  BIERGARTEN #24
  CAFE #25
  FASTFOOD #26
  ICECREAM #27
  PUB #28
  RESTAURANT #29
```

Figure 8: Representations of the same classification scheme: (*left*) in CSV format, (*right*) in YAML format as used in TripleGeo ver.1.4

3.3.4. Transformation Modes

As already mentioned, TripleGeo ver.1.4 offers four different *modes* for transformation of geospatial features to RDF:

1. *GRAPH* mode, which is disk-based and was the only option supported by ver.1.1 of the software;
2. *STREAM* mode, which work in main memory was introduced with ver.1.2;
3. *RML* mode also works in memory and introduced with ver.1.3;
4. *XSLT* mode is used only in transformation of semi-structured data, and has not been modified since its launch with ver.1.1.

Next, we provide more details about the specifics of each transformation mode.

3.3.4.1. GRAPH Transformation Mode

This transformation mode employs a Jena *Model* [JenaDoc] for collecting all RDF triples produced during transformation and before emitting them to a file. This Model denotes an *RDF graph*, so called because it contains a collection of RDF nodes attached to each other by labelled relations. The Jena Model has a rich API with many methods intended to make it easier to write RDF-based programs and applications. Besides, a Model also provides an abstraction over different ways of storing the RDF nodes and relations: in-memory data structures, disk-based persistent stores and inference engines. In contrast to the initial release ver.1.1

of TripleGeo where models were retained in main memory, in subsequent versions (including the current ver.1.4) we always employ *disk-based* models when the GRAPH transformation mode is chosen. This is a deliberate decision, in order to account even for very large POI datasets that may not be fully accommodated in main memory, thus incurring excessive computational cost for transformation. This persistent graph essentially stores custom disk-based tuple indices regarding the triples obtained after transforming each input feature (i.e., a POI). In case new triples are being added to the store, the respective indices need to be updated.

In Jena, a single triple is represented as a *Statement* with a *Subject*, a *Predicate*, and an *Object*. According to the RDF specification, only resources can be the subject of an RDF triple, whereas the object can be a resource or a literal. Predicates express the relationship between nodes according to a given ontology; in our case, this is the SLIPO ontology [SLIPO-D2.1] for POIs. Jena models support two distinct types of nodes: *URI* references and *literals*: the former denotes resources for which some assertions are made, whereas the latter denote concrete data values that appear in those assertions. A resource represented as a URI denotes a named thing. In SLIPO, this refers to a POI that has a distinct identity (e.g., a unique identifier in the dataset), which can be used as a direct reference to that resource. Literals representing values other than strings may have an attached *data type*, which helps an RDF processor convert the string representation of the literal into the correct value.

Apart from numeric values, dates, etc. that can be represented by suitable XSD data types, in SLIPO there is the crucial requirement to support *geometries* of POIs. Such geometries denote the location of the POI and can be not only points (i.e., a pair of longitude/latitude coordinates for a restaurant), but also linestrings (e.g., for a scenic route POI), polygons (e.g., the spatial extent of an archaeological site), or even more complex geometries (e.g., a geometry collection with the area, the boundary and the centroid of a building). Such geometry literals are supported according to the GeoSPARQL standard [OGC12], even though the Jena model is *agnostic* of geospatial data types and operations. For instance, topological queries like "find the POI at a given location" (specified by coordinates) cannot be answered with a SPARQL query against the RDF graph in Jena. However, these geometries can be retained in the model and TripleGeo can correctly emit them to the output RDF files with proper WKT serialization according to GeoSPARQL standard.

When the entire input dataset is consumed and its contents have been transformed into statements in the RDF graph, TripleGeo exports the model into a file according to a user-specified serialization format containing all triples in the graph.

3.3.4.2. STREAM Transformation Mode

Jena supports processing operations over RDF in a streaming fashion [JenaStream] in cases that applications need to manipulate RDF data at scale. High performance readers and writers for all standard RDF formats are available with the Jena RIOT (RDF I/O technology) API, also extensible with custom formats. N-Triples/N-Quads provide the highest input parsing performance using W3C Standards.

In transforming geospatial data into RDF with the STREAM transformation mode in TripleGeo, we are actually concerned with writing RDF data as a stream. Unfortunately, not all RDF formats are suitable for streamlined writing to files. N-Triples and N-Quads are always written as a stream. Formats that provide pretty printing (for example the default RDFFormat for each of Turtle, TriG and RDF/XML) require analysis of

the entire model in order to determine nestable structures of blank nodes and for using specific syntax for RDF lists.

So, for the STREAM transformation mode in TripleGeo, we employ a different strategy. We handle each incoming feature (i.e., a POI with its geometry and all its thematic attributes) in isolation from the rest and we can transform it to triples according to the given YAML mapping to the ontology. Of course, this is implicitly based on the reasonable assumption that all information about a POI is fully included *in a single record*, and there are no properties that need be searched in other entities (e.g., by joins to other input tables or files). This is the typical case in almost all POI datasets available and certainly all those handled in the context of SLIPO. So, when all attribute values of a given POI have been turned into triples, these can be readily emitted and written to the output file. The only restriction is their *serialization*, because they can only be written in N-Triples/N-Quads as a stream.

Regarding writing to output, the user may regulate the rate at which results are stored in the files in order to avoid high I/O interaction with the disk and reduce execution time. More specifically, in the TripleGeo configuration file, the user can prescribe a number of input records that will be transformed in the same *batch* (by default, a batch consists of 10 records), so that their resulting triples are first accumulated in memory and spilled to the disk file together. Ideally, the size of the batch represents a trade-off between not too frequent disk I/O and moderate size of collections of transformed triples retained in memory.

By not maintaining a persistent graph and promptly emitting transformed triples, this STREAM mode can achieve orders of magnitude faster execution, as reported in our experimental results in Section 6.

3.3.4.3. RML Transformation Mode

In order to be able to invoke this transformation mode with TripleGeo, suitable RML mappings for input data features should have been specified in advance (as explained in Section 3.3.2.1) and declared in the configuration file.

In aligning RML to work with TripleGeo, we have made slight modifications in source code regarding RML processing. More specifically, an instance of *RML Processor* is created, equipped with as many *RML performers* as the Triples Map constructs defined in the RML mappings. The RML processor extracts in advance all iteration patterns corresponding to *Triples Maps* constructs in the RML mappings. Hence, each Triples Map activates a dedicated *RML Performer* that is employed to handle specific fragments of an input feature (e.g., specific thematic attributes of a POI regarding its postal address).

In terms of integration within TripleGeo, we have modified RML processing to work in a *streaming* fashion. Note that the original software [RML], consumes input data and creates a materialized Sesame SAIL (Storage and Inference Layer) repository where all transformed triples are stored, and finally exports all output to a user-specified serialization. As this dependence on materialized (in memory or on disk) repositories may easily become a bottleneck in transforming large datasets, we created a wrapper over RML processors for iterating over each input feature and producing transformed triples according to the RML mappings. Hence, for each incoming feature provided by the Feature Iterator over the input data, all activated RML Performers are applied. The defined Subject Map and Predicate-Object Maps are applied against attribute values and the corresponding triples are generated. When necessary, execution of dependent Triples Map is triggered by the appropriately defined *Parent Triples Map* and a *nested mapping* is being applied. All generated triples

are kept in an in-memory collection and not materialized on disk. As in the STREAM mode, the user may specify a *batch* size for writing the output in order to minimize I/O cost, so that collected triples are only written to the output file periodically, i.e., once a given number of input records have been processed. Again, as in the STREAM mode, the only restriction is *serialization* of triples, because they can only be streamlined in N-Triples/N-Quads format to the output file.

3.3.4.4. XSLT Transformation Mode

This mode can be *exclusively* used for transforming *semi-structured* geographical files into RDF. In particular, it accepts input datasets in GML, KML, and XML formats, as well as *INSPIRE-aligned* data (GML) and metadata (XML). *Geography Markup Language* (GML) is an OGC standard [GML] for representing geospatial information. The basic primitives of GML are spatial features as locations on Earth, and their geometric shapes are modelled with vectors (points, lines, polygons, etc.). The original GML model (version 1.0) was based on RDF/RDFS profiles, but afterwards the OGC introduced XML schemas for interoperability with existing spatial databases. Although this RDF profiling is no longer supported, subsequent GML models (current version is 3.3) still retain certain features of RDF, and most importantly the concept of child elements as properties of a parent resource (RDFS). Thanks to this relaxed "triple-like" binding, GML (as well as similarly structured KML and XML) features may be transformed into RDF using XSLT transformation.

In practice, the user needs to create an application profile for the input dataset taking advantage of a set of predefined *templates* concerning mappings of geometries and thematic attributes for XSLT transformation. Once invoked, such a custom *XSL style sheet* accepts an XML, GML or KML file and maps each input feature into suitable RDF statements according to the mapping. The result is an RDF/XML representation of input GML and KML data or XML metadata. Output RDF files can be readily loaded into a triple store.

TripleGeo also supports *INSPIRE-aligned data* (in GML) and *metadata* (in XML). Regarding such XML metadata, a generic XSL stylesheet (`Metadata2RDF.xsl`) covers all elements and can be reused against any metadata conforming to INSPIRE specifications. Besides, we have introduced one custom XSL stylesheet for each INSPIRE Data Theme (Annex I) [INSPIRE-Themes], practically translating each domain-specific element in the respective GML schema into a suitable RDF resource. Using XSL stylesheets for GML representations of INSPIRE-compliant features seems most preferable and generic, as these scripts can be re-used and can work with any XSLT parser. Regarding handling of 2-dimensional OGC geometries, a generic stylesheet (`GML2WKT.xsl`) can convert from Geometry Markup Language (GML) to Well Known Text (WKT) representations according to the GeoSPARQL standard [OGC12]. This script can cope with a wide range of both primitive and complex geometric types, including identification of the spatial reference system (CRS) of every geometry in order to provide a complete WKT representation for the resulting RDF dataset.

For convenience, all these XSL style sheets have been integrated into TripleGeo since ver.1.1. Users may edit existing style sheets in order to define suitable values for the attributes, namespaces, and other specifications regarding the data or metadata at hand, and then perform invoke TripleGeo to perform an XSLT transformation and finally obtain the resulting RDF files.

3.3.5. Identifiers Registry

TripleGeo can support the process of registering POI data to the Identifiers Registry. The Identifiers Registry is a component of the SLIPO architecture that serves a twofold purpose:

- *Creation and assignment of identifiers*. The Identifiers Registry includes a configurable mechanism for creating and assigning identifiers to POIs. As described in Section 3.3.1, TripleGeo offers functionality for assigning URIs to POIs during the transformation process. However, in certain cases, it may be desirable or required to decouple the identifier generation process from the transformation process (e.g., if the user wishes to apply a different identifier scheme than that supported by TripleGeo, or in case transformation is handled by another tool that does not generate identifiers according to the expected scheme). In such cases, the user can rely instead on the Identifiers Registry to generate and assign an identifier (URI) to each POI, which substitutes any temporary identifier that may have been assigned to it during transformation.
- *Lookup service for POIs*. The Identifiers Registry maintains a database associating each registered POI with its respective identifier (URI), which may be either the one assigned to it during transformation or the one generated by the Identifiers Registry itself. In addition, it stores certain basic attributes about each registered POI, such as its name, category and location (see Section 3.2.3). Thus, the Identifiers Registry can serve as a lookup service for retrieving basic information about a previously imported POI or for checking whether a given POI has already been registered, and which identifier has been assigned to it.

Interaction with the Identifiers Registry takes place via a RESTful API. Next, we list the supported API calls.

3.3.5.1. POI Registration

Registration of new POIs can be done in batch mode using the following method:

https://[registry.dev.slipo.eu]/register	(Request type: POST)

The method accepts a list of POIs to be registered, where each entry in the list contains the following parameters:

- *poi_id* (optional). If present, the POI is registered with the specified id, otherwise the Identifiers Registry generates and assigns an identifier to the POI.
- *source*. An identifier specifying the source of this POI.
- *source_id*. The identifier of the POI in its source.
- *names*. A list of known names for the POI.
- *categories*. A list of categories associated with the POI.
- *geom*. The (default) geometry of the POI in GeoJSON format.

The response is a JSON document containing the following information:

- *success*. A Boolean value indicating whether an error has occurred.
- *errors*. In case of an error, an array containing messages for the errors that occurred.

- *result*. In case of success, an array of objects, each one contained the source_id of the POI that was registered and the id that was assigned to it (in case of relying on the Identifiers Registry to assign a new id to the POI).

3.3.5.2. POI Deletion

An existing POI entry can be removed from the Identifiers Registry using the following method:

https://[registry.dev.slipo.eu]/delete	(Request type: POST)

Input parameters:

- *poi_id*. The identifier of the POI to be deleted.

Response:

- *success*. A Boolean value indicating whether an error has occurred.
- *errors*. In case of an error, an array containing messages for the errors that occurred.

3.3.5.3. POI Search

The following methods are provided for retrieving POI entries from the Identifiers Registry.

https://[*registry.dev.slipo.eu*]/poi/id_search	(Request type: POST)

This method retrieves POI entries by their id.

Input parameter:

- *poi_id*. The URI used to identify the POI.
- *categories* (optional). A list of categories used for filtering.

Response:

- *success*. A Boolean value indicating whether an error has occurred.
- *errors*. In case of an error, an array containing messages for the errors that occurred.
- *result*. A POI entry (comprising the basic information stored in the Identifiers Registry about the POI).

https://[*registry.dev.slipo.eu*]/poi/source_id_search	(Request type: POST)

This method retrieves POI entries by their source and source id.

Input parameter:

- *source*. An identifier specifying the source of the searched POI.
- *source_id*. The identifier of the POI in its source.

Response:

- *success*. A Boolean value indicating whether an error has occurred.
- *errors*. In case of an error, an array containing messages for the errors that occurred.
- *result*. A POI entry.

https://[*registry.dev.slipo.eu*]/poi/name_search	(Request type: POST)

This method retrieves POI entries by name.

Input parameters:

- *name*. The name of the POI.
- *categories* (optional). A list of categories used for filtering.

Response:

- *success*. A Boolean value indicating whether an error has occurred.
- *errors*. In case of an error, an array containing messages for the errors that occurred.
- *result*. An array of POI entries.

https://[*registry.dev.slipo.eu*]/poi/bbox_search (Request type: POST)

This method retrieves POI entries by a bounding box.

Input parameters:

- *left*. The minimum x-coordinate.
- *right*. The maximum x-coordinate.
- *bottom*. The minimum y-coordinate.
- *top*. The maximum y-coordinate.
- *categories* (optional). A list of categories used for filtering.

Response:

- *success*. A Boolean value indicating whether an error has occurred.
- *errors*. In case of an error, an array containing messages for the errors that occurred.
- *result*. An array of POI entries.

https://[*registry.dev.slipo.eu*]/poi/radius_search (Request type: POST)

This method retrieves POI entries within a given radius from a given point.

Input parameter:

- *x*. The x-coordinate of the point.
- *y*. The y-coordinate of the point.
- *radius*. The radius of the search.
- *categories* (optional). A list of categories used for filtering.

Response:

- *success*. A Boolean value indicating whether an error has occurred.
- *errors*. In case of an error, an array containing messages for the errors that occurred.
- *result*. An array of POI entries.

3.4. Deployment on Multiple Worker Threads

As already discussed, each POI is considered as a separate, autonomous entity with its own properties (name, category, locations, etc.), e.g., represented as a single record in a Shapefile or DBMS table. This is the most typical representation of a POI in most data sources. Thus, when it gets transformed into RDF, currently TripleGeo generates a collection of triples per POI without any links to other POIs. Of course, it may occur that a POI could refer to another POI for expressing *relationships* (e.g., a shop within a shopping centre, an office in a building), but such cases will be dealt with in future releases of TripleGeo.

Assuming that such conceptual isolation of information concerning each individual POI holds in an input dataset, it opens up opportunities for advanced performance in their transformation to RDF and scalability with large data volumes. Quite simply, a POI dataset may be split into a number of *disjoint subsets*, each one ideally having an equivalent number of POIs. *Data partitioning* may be carried out in several fashions, e.g., spatially using a uniform grid and exporting POIs within a cell into a separate file, or by a specific attribute (e.g., per category or country of origin), although such methods may yield subsets with varying size. But the key idea is that each chunk may be processed separately and yield its own output of transformed RDF triples. Finally, these partial results may be merged into a single one, or loaded into a triple store to create a unified RDF representation for the original data.

We have adopted this strategy in TripleGeo since ver.1.2, by employing a *number of threads* to cope with disjoint data chunks. In particular, the user may specify in the configuration any number of input datasets, provided that they all comply to the same attribute schema and the same file format. Currently, TripleGeo neither imposes nor implements any particular data partitioning scheme, but it accepts already pre-processed subsets of data. For each such chunk, a separate thread of TripleGeo is launched for its transformation, which proceeds totally isolated from the rest. Each thread abides by the same configuration settings, e.g., applies the same classification scheme, attribute mappings, namespaces, etc., as if it worked alone. Transformation is completed once all threads conclude their task, and resulting triples are written into separate files (one per thread) corresponding to its subset of the input data. Depending on the specified serialization, there is the option to merge output files into a unified one (e.g., in case that N-Triples is the chosen serialization).

As our experiments is Section 6 testify, employing multiple concurrent threads for transforming disjoint pieces of large POI datasets can offer orders of magnitude performance gains. Of course, this partitioning is currently prepared in advance, without any intervention from TripleGeo. In the immediate future, we will include data partitioning strategies into TripleGeo in order to make estimates about suitable partitions without user interaction, taking into account available system resources (CPU, memory, disk), as well as statistics over data characteristics (spatial distribution, number of attributes, etc.).

4. Reverse Transformation from RDF

As already mentioned, transformation is actually a *bidirectional* process that should also allow the backward transformation of linked POI data (*potentially interlinked or fused*) into conventional POI formats, and thus enabling existing products, systems, and services to exploit the integrated POI datasets. TripleGeo supports this reverse transformation of RDF POI data into de facto POI formats (*currently, CSV and ESRI shapefiles*). Of course, there exists an impedance mismatch in this direction, given that the SLIPO ontology is semantically more expressive than the conventional POI schemata. Presently, in TripleGeo ver.1.4 we have prepared specifications (in SPARQL) that generally retrieve the same attributes as in original geospatial files given to transformation. But generally, POI attributes, relations and metadata in RDF representation will be richer than what can be supported by conventional file formats. In the future, we will consider more advanced reverse transformations that will allow incorporation of the maximum possible amount of semantic (linked, enriched, fused) POI information and metadata into the resulting attributes of conventional POI formats.

In this Section, we outline the processing flow of reverse transformation and explain how it is possible to reconstruct RDF data on POIs with geometries into records stored in a geospatial file.

4.1. Architecture

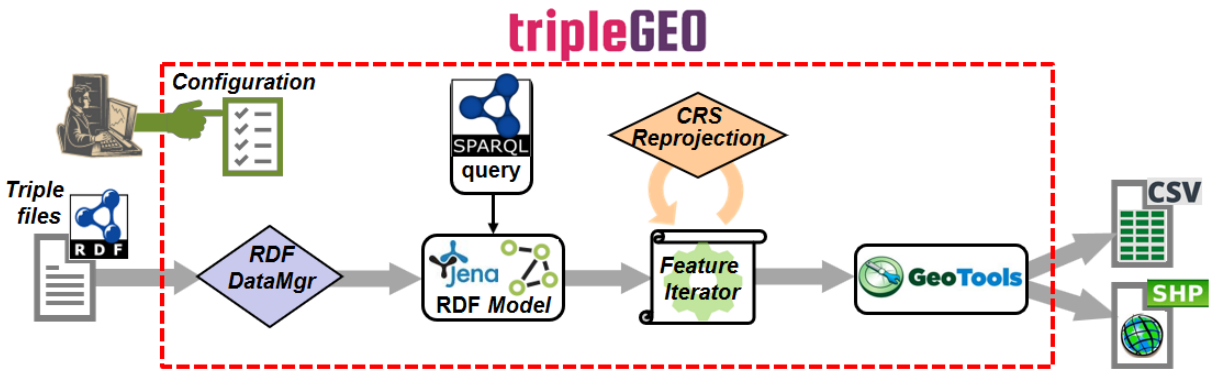


Figure 9: Processing flow of reverse transformation from RDF in TripleGeo ver.1.4

As in the case of transformation to RDF, the *reverse transformation* functionality of TripleGeo ver.1.4 works in a straightforward fashion according to some preconfigured settings. Figure 9 illustrates the flow diagram used for reconverting RDF triples with geometries into records in a geospatial file. Next, we outline the basic components of this utility:

Input RDF data may be obtained from files with standard RDF serializations. Multiple RDF files may be specified (with the same serialization and with statements obeying the same ontology) in order to reconstruct a *single* geospatial file with all (geometric and thematic) information.

A *configuration file* lists all properties that are used to control the various stages of reverse transformation: how input data will be accessed, which data is involved, 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 5.2.2.

TripleGeo ver.1.4 achieves reverse transformation from RDF to geospatial format by creating an intermediate disk-based Jena *model* that stores all input triples (equivalent to the *GRAPH* mode in transformation). Hence, the native *RDFDataMgr* available from Jena API is applied against the input triple files in order to build this RDF model.

The user must specify (in a separate file) a SELECT query in SPARQL that will be used to retrieve results (records) from the constructed model. In order to execute correctly and return meaningful results, this query should conform with the underlying ontology of the input RDF triples.

Once the query is submitted and results are returned, a *feature iterator* consumes each result and recreates a record from it. In particular, each attribute concerning a POI and specified in the SELECT query becomes a column in the resulting file. Optionally, *reprojection* of geometries into another spatial reference system (CRS) is available. This coordinate transformation is carried out thanks to the integrated GeoTools library and according to user specifications for the source and target CRS.

Regarding *thematic* (i.e., non-spatial) attribute values (e.g., type, name, contact information) of an input feature, these are generally stored as strings. This is typical when output is written into *CSV* files. With respect to writing output to *ESRI shapefiles*, the integrated GeoTools library provides some functionality for defining data types. So, in case that the data type (e.g., date or numeric) is specified for a value in the RDF input, the respective attribute is defined with the equivalent type available in the shapefile format; otherwise, these are stored as strings.

Finally, note that reverse transformation in TripleGeo ver.1.4 does not support parallelization, but works on a *single thread* instantiation only. Even in the case of multiple input RDF files, these are acquired one by one and inserted into a common RDF model so as to generate a single output file containing all reconstructed geospatial features. In future releases, we will consider parallelization of the reverse transformation process.

4.2. Input and Output

4.2.1. Input

In terms of input RDF serializations, and according to the specifications of the Jena API [JenaDoc] that is used to build the disk-based model, triples can be obtained in one of the following formats:

- RDF/XML;
- RDF/XML-ABBREV;
- N-TRIPLES;
- N3;
- *TURTLE* (also abbreviated as *TTL*).

Note that these serializations are exactly the same ones supported for the output RDF files produced by the transformation functionality of TripleGeo. In terms of standardization, the input triples are conformant to

W3C standards, in order for the compiled model to be queryable via SPARQL. With respect to geometries used as objects in triples, these should be specified as WKT literals according to the GeoSPARQL standard [OGC12]. In the future, we aim to also support point geometries consisting of a pair of triples (one for longitude, another for latitude), as specified in legacy vocabularies like Virtuoso [VirtGeoRDF] or WGS84 RDF Geoposition [GeoPos84].

4.2.2. Output

Results from the reverse transformation by TripleGeo can be stored in one of the following *vector file formats*:

- ESRI *shapefiles* [ESRIshp]. Note that only one of the primitive geometric data types for (Multi)Points, (Multi)LineStrings, and (Multi)Polygons is allowed in a single shapefile. In addition, data types for thematic attributes are converted to those supported by shapefiles, e.g., timestamp values must be converted to strings in order to be accepted.
- *CSV* containing geometries either as pairs of (x,y) coordinates (for points) or in WKT serialization (for any geometry type). Users must specify a suitable delimiter character (not present in any attribute value), and optionally a quote character for enclosing string values.

5. TripleGeo Usage Manual

In this Section, we provide a complete usage manual for TripleGeo ver.1.4. First, we give details on building the application from the Java source code, along with particular details on certain of its dependencies. Next, we describe configuration settings both for transformation to RDF and for reverse transformation. Finally, we provide execution examples that demonstrate its operation and indicate certain limitations of the software mostly related to specific platforms.

5.1. Building Installation

Version 1.4 of TripleGeo, as well as all its previous releases, are publicly available, offering the entire Java source code as well as indicative configurations [TripleGeo]. TripleGeo is a command-line utility and has several dependencies on open-source and third-party, freely redistributable libraries.

Java SDK 1.8 (or later) as well as Maven 3.5.0 (or later) [MAVEN] must be installed and properly configured in order to compile and execute TripleGeo. The `pom.xml` file contains the project's configuration in Maven and has been successfully tested in both MS Windows and Linux environments.

Special note on JDBC drivers for database connections: In case that data will be extracted from a geospatially-enabled DBMS (e.g., PostGIS), either the user must include the respective .jar (e.g., `postgresql-9.4-1206-jdbc4.jar`) in the classpath at runtime or to specify the respective dependency in `pom.xml` and then rebuild the application.

Special note on manual installation of a JDBC driver for Oracle DBMS: Due to Oracle license restrictions, there are no public repositories that provide `ojdbc7.jar` (or any other Oracle JDBC driver) for enabling JDBC connections to an Oracle database. You need to download it and install in your local repository. First, this jar must be downloaded from Oracle and then installed it in a local maven repository as follows:

```
mvn install:install-file -Dfile=/<*YOUR_LOCAL_DIR*>/ojdbc7.jar -
DgroupId=com.oracle -DartifactId=ojdbc7 -Dversion=12.1.0.1 -Dpackaging=jar
```

Starting from version 1.3, TripleGeo includes support for custom transformation of thematic attributes according to the RDF Mapping language [RML]. In order to enable RML transformation mode, the respective library `RML-Mapper.jar` specially prepared for TripleGeo execution must be installed in a local Maven repository as follows:

```
mvn install:install-file -Dfile=/<*YOUR_LOCAL_DIR*>/RML-Mapper.jar -
DgroupId=be.ugent.mmlab.rml -DartifactId=rml-mapper -Dversion=0.3 -
Dpackaging=jar
```

Building the application with Maven can be done as follows:

```
mvn clean package
```

and results into a `triplegeo-1.4-SNAPSHOT.jar` under directory target according to what has been specified in the `pom.xml` file.

The current distribution (ver. 1.4) comes with dummy configuration templates `file_options.conf` for geographical files (ESRI shapefiles, CSV, GPX, KML, etc.) and `dbms_options.conf` for database contents (from PostGIS, Oracle Spatial, etc.). These files contain indicative values for the most important properties when accessing data from geographical files or a spatial DBMS. Self-contained brief instructions guide the user into the extraction and transformation process.

Configuration files for several cases and file formats are available in the SLIPO GitHub [TripleGeo] in order to guide the users into building their own. Indicative configurations are also given in the Annex (Section 9.2).

5.2. Configuration Settings

5.2.1. Configuration for Transformation to RDF

Transformation mode
mode

Export format
serialization

Spatial Ontology
targetGeoOntology

Input/Output
inputFormat
encoding
batchSize
inputFiles
tmpDir
outputDir

CSV specifications
delimiter
quote

Mapping specification
mappingSpec

Spatial Reference Systems
sourceCRS
targetCRS

Classification scheme
classificationSpec
classifyByName

Data Source and Namespaces
featureSource
nsOntology
nsGeometry
nsFeatureURI
nsClassificationURI
nsClassURI
nsDataSourceURI
prefixes
namespaces

DBMS connection and data details
dbType
dbName
dbUserName
dbPassword
dbHost
dbPort
tableName
filterSQLCondition

Basic attributes
attrKey
attrGeometry
attrX
attrY
attrName
attrCategory
registerFeatures

Other properties
defaultLang

Table 2: Properties specified in configuration files for transformation to RDF using TripleGeo ver.1.4

Before attempting any transformation using TripleGeo, a configuration file must be prepared. This file lists several *mandatory* 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,

classification scheme for assigning categories to features). The list of all properties that can be specified for transformation into RDF is shown in Table 2. An indicative configuration file that has been applied to transform data from an ESRI shapefile into RDF is provided in the Annex (Section 9.2.1). Some of the basic properties in this configuration are listed in Figure 10.

```
inputFormat = SHAPEFILE
inputFiles = ./test/data/points.shp
encoding = UTF-8

tmpDir = ./tmp
outputDir = ./test/output
serialization = N-TRIPLES

targetGeoOntology = GeoSPARQL

mappingSpec = ./test/poi_mappings.yml

classificationSpec = ./test/poi_classification.csv

mode = STREAM

attrKey = osm_id
attrGeometry = the_geom
attrName = name
attrCategory = type

nsOntology = http://slipo.eu/def#
nsGeometry = http://www.opengis.net/ont/geosparql#
nsFeatureURI = http://slipo.eu/id/poi/

sourceCRS = EPSG:2100
targetCRS = EPSG:4326
```

Figure 10: Example of a configuration file as used in TripleGeo ver.1.4

Next, we provide full details about all properties that can be specified in this *configuration file*:

1. ***Transformation mode***. The user must specify *mandatory* property

   - *mode*: Controls the execution mode for the transformation and can take one of the following values: *GRAPH* (using a disk-based Jena model), *STREAM* (in-memory conversion with prompt creation of triples per input feature), *RML* (for applying user-specified RML schema mappings), or *XSLT* (for handling XML/GML/KML/INSPIRE-aligned input with XSLT transformation).

2. Input and output properties.

   - *inputFormat*: This *mandatory* property specifies the format of the input data. In case of *geographical file(s)*, supported vector data formats include: *SHAPEFILE*, *CSV*, *GPX*, *GEOJSON*, *XML* (for handling XML/GML/KML/INSPIRE-aligned input), and *OSM* (currently, only supporting OpenStreetMap XML format). In case that input features reside in a *database*, value *DBMS* must be specified for this parameter.

   - *encoding*: This optional property specifies the encoding (character set) for strings in the *input* data, such as UTF-8, ISO-8859-1, ISO-8859-7, WINDOWS-1253, etc. If not specified, UTF-8 encoding is the considered as the default. Note that string literals in *output* RDF triples are always in UTF-8.

- *batchSize*: This *optional* parameter indicates the number of input features to transform in each batch before storing output to file. This is done for performance, since too frequent writes to disk may slow down the speed of transformation. By default, the current version spills output triples to file once a batch of 10 input records is processed.

The user must always specify (relative or absolute) *paths* to directories and files used during processing:

- *inputFiles*: The user may specify *multiple* input files (of exactly the same format and attribute schemata) separating them by ';' in order to activate multiple concurrent threads for their transformation. Specification of input files should be omitted in case that data resides in a geospatial DBMS (there is another parameter concerning the *tableName* that should be specified instead).
- *tmpDir*: The working directory that may be used for storing intermediate files temporarily created during transformation (e.g., the Jena model created on disk with the GRAPH transformation mode).
- *outputDir*: Directory where the output RDF file(s) will be stored. By default, the output file name(s) are automatically composed from the original input file name(s) with the extension of the respective RDF serialization, e.g., *points.nt*. Files with the same name previously created in the output directory will be overwritten.

In case of CSV input, two additional parameters must be specified (omitted for any other input formats):

- *delimiter*: designates the character delimiting attribute values for each input line (i.e., record). This single character must not appear in any attribute value.
- *quote*: designates the quote character for enclosing string values in attributes.

3. ***Export format*** for the output file(s) can be specified with mandatory property:

- *serialization*, which can be set to one of the following values: *RDF/XML* (used as default for the XSLT transformation mode), *RDF/XML-ABBREV*, *N-TRIPLES* (used as default for the STREAM and RML transformation modes), TURTLE (or TTL), and N3.

4. ***Spatial ontology*** for geometries in the exported RDF data. This depends on the triple store where the exported data will be imported (e.g., Virtuoso, Oracle), since geometric representation and geospatial support varies widely amongst them. This must be defined in mandatory property

- *targetGeoOntology*, which can currently support three possible options: *GeoSPARQL* (default) for subsequent import to compliant triple stores (e.g., Oracle, Parliament), *Virtuoso* for extracting point features using the legacy RDF ontology only for points in Virtuoso RDF (namespace *virtrdf:*) [VirtRDF] or *wgs84_pos* for point features under the WGS84 Geoposition RDF vocabulary [GeoPos84].

5. ***Mapping specification***. The current version of TripleGeo supports mappings from the input attribute schema into an ontology for RDF features that guides the transformation (i.e., creating RDF properties, constructing URIs, defining links between entities, etc.). These mappings will be utilized in transformation once configuration property

- *mappingSpec* specifies the (absolute or relative) path to a file (in TTL, YAML, or XSL format) that contains these mappings.

Such mappings can be defined in three alternative file formats and employed in diverse transformation modes:

i. In *RML* transformation mode, a TTL file (in TURTLE format) contains RML mappings from input schema to RDF. In RML mode, specifying mappings with this file is *mandatory*, otherwise no RDF triples will be produced.

ii. In *GRAPH* or *STREAM* transformation modes, a file (in YAML format) contains mappings from input schema to RDF according to a custom ontology (such as the SLIPO ontology for POIs [SLIPO-D2.1]). In GRAPH/STREAM modes, this parameter is *optional*; if left blank or omitted, then an RDF property will be created for each thematic attribute in the original schema, by borrowing its attribute name.

iii. In *XSLT* transformation mode, a XSL style sheet file determines the XSL schema mapping for attributes to be converted. Specifying such a file is *mandatory* for XSLT transformation, otherwise no RDF triples will be produced.

6. ***Classification scheme***. Optionally, classification of input features into categories can be also performed during transformation, provided that the user specifies a (possibly hierarchical, multi-tier) classification scheme (e.g., possible amenities for Points of Interest, a list of road types for a Road Network). Classification is only applied if a suitable *mappingSpec* (including a category attribute) has been also specified. This classification scheme can be prescribed with these configuration properties:

- *classificationSpec*: File (either in CSV or YAML format) containing a classification hierarchy in categories assigned to input features. This property should be left blank in case of no applicable classification scheme.
- *classifyByName*: Boolean parameter indicating whether the data features specify their category based on its identifier in the classification scheme (*false*) or the actual name of the category (*true*). By default, transformation uses identifiers of categories in the classification scheme. This parameter is ignored if no classification hierarchy has been properly defined (i.e., missing or wrong path in parameter *classificationSpec*).

7. ***DBMS connection and data details***. The following properties are *mandatory* when connecting to a DBMS and extracting features from a spatial table. In case that any other value has been specified in parameter *inputFormat*, these parameters should be omitted altogether.

- *dbType*: Specify the DBMS backend where spatial data is stored. Possible values: *MSAccess*; *MySQL*; *Oracle*; *PostGIS*; *DB2*; *SQLServer*; *SpatiaLite*.
- *dbName*: Name of the database to connect. For MS Access databases, specify absolute or relative path to the .mdb database file.
- *dbUserName*: Username for JDBC connection. For MSAccess databases, credentials are optional; must be specified only if required to access the .mdb database file.
- *dbPassword*: Password for JDBC connection. For MSAccess databases, credentials are optional; must be specified only if required to access the .mdb database file.

- *dbHost*: The host name or IP address on which DBMS server listens for TCP/IP connections from client applications. Value *localhost* should be set if working with a local database server. Omit for MSAccess database connections.
- *dbPort*: Specify the TCP/IP port on which the DBMS server listens for connections from client applications. Omit for MSAccess database connections.

In addition, the following two properties specify which data to extract from the database:

- *tableName*: *mandatory* property that indicates the database table or view which contains the spatial features to be extracted.
- *filterSQLCondition*: *optional* property that specifies a filter for selecting qualifying records with syntax equivalent to a `WHERE` clause in SQL, e.g., `town_type = 'TOWN' OR town_type = 'VILLAGE' OR town_type = 'CITY'`. 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 this condition is left blank, no filter applies and all records in the table will be extracted.

8. ***Basic attributes*** that characterize each input feature and may be optionally used for registering it into the SLIPO Registry. Those basic attributes can be specified with the following properties:

- *attrKey* : mandatory column name containing unique identifier for each feature (i.e., each record). Until TripleGeo ver.1.3, this was required in order to guarantee suitable, unique URIs. Starting from TripleGeo ver. 1.4, a unique key for each input record is no longer required, since URIs are based on UUIDs, which are generated on-the-fly. However, it should be recommended that input features have a unique key in order to avoid duplicate triples with the same input contents.
- *attrGeometry*: specifies the name of the geometry column in the input table or file. Omit this parameter if geometry representation is available with columns specifying pairs of *X,Y* coordinates for points; otherwise, this parameter is mandatory.
- *attrX*: Specifies the attribute holding *X*-coordinates (or longitude) of point locations. Mandatory if a geometry attribute has not been specified above; otherwise, this property may be omitted.
- *attrY*: Specifies the attribute holding *Y*-coordinates (or latitude) of point locations. Mandatory if a geometry attribute has not been specified above; otherwise, this property may be omitted.
- *attrName*: optional property that specifies the column name containing name literals (i.e., strings). By default, NULL values in this attribute are suppressed and are not exported in order to avoid blank nodes. Applicable when name values are used for registering features in the SLIPO Registry. Leave blank if non applicable.
- *attrCategory*: optional property that specifies the column name containing literals regarding classification into categories (e.g., type of points, road classes etc.) for each feature. Values in this attribute will be joined with the classification scheme specified in order to generate links of output entities with category URIs. By default, NULL values in this attribute are suppressed and are not exported in order to avoid blank nodes. Applicable when category values are used for registering features in the SLIPO Registry. Leave blank if non applicable.

Such basic attribute values per feature along with its assigned URI (generated on-the-fly during transformation) can be exported from TripleGeo into a separate CSV file in order to be used in the *SLIPO Registry*. This is controlled by property

- *registerFeatures*: an *optional* Boolean parameter that denotes whether a .CSV file will be also extracted (*true*) or not (*false*) specifically for registering features in the SLIPO Registry. This property should be omitted in all other cases.

9. Data Source and Namespace properties.

- *featureSource* is a mandatory string value (e.g., OpenStreetMap) that specifies the data source provider of the input features in order to include this information in the resulting triples for each entity. Unlike previous versions of TripleGeo, in ver.1.4 this value is no longer used in the assignment of URIs to features.

The namespaces and prefixes for the utilized ontology and the resources that will be generated are set with the following optional properties:

- *nsOntology*: the namespace of the underlying ontology. Used in creating properties for the RDF triples., e.g., http://slipo.eu/def# for the SLIPO ontology (*default*).
- *nsGeometry*: the namespace for the underlying geospatial ontology, e.g., http://www.opengis.net/ont/geosparql# for GeoSPARQL-compliant geometries (*default*).
- *nsFeatureURI* : the common URI namespace for all generated resources, e.g., http://slipo.eu/id/poi/ in the SLIPO ontology (*default*).
- *nsClassificationURI*: the common URI namespace for the classification scheme, e.g., http://slipo.eu/id/classification/ in the SLIPO ontology (*default*).
- *nsClassURI*: the common URI namespace for categories used in the classification scheme, e.g., http://slipo.eu/id/term/ in the SLIPO ontology (*default*).
- *nsDataSourceURI*: the common URI namespace for the data source provider, e.g., http://slipo.eu/id/poisource/ in the SLIPO ontology (*default*).

In addition, the user may also define two lists (of comma separated values) with the correspondence between a prefix and its respective namespace (mainly used in attribute mappings):

- *prefixes*: A list of prefixes employed in the (RML or YAML) mapping files, e.g., *slipo*, *geo*, *rdfs* .
- *namespaces*: the namespaces corresponding to the aforementioned prefixes, e.g., http://slipo.eu/def#, http://www.opengis.net/ont/geosparql#, http://www.w3.org/1999/02/22-rdf-syntax-ns# .

10. ***Spatial Reference Systems***. If geographic reprojection is required for geometries in the output triples, then the following properties must be filled in the configuration:

- *sourceCRS*: the EPSG identifier for the coordinate reference system (CRS) of the input geometries;
- *targetCRS*: the EPSG identifier for the coordinate reference system (CRS) of the output geometries.

In case that *either* of these properties is missing, the respective geometries are assumed to be in the WGS84 reference system (EPSG:4326). TripleGeo works for any valid EPSG reference systems [EPSG] and transforms all geometries in the dataset, e.g., from *sourceCRS*=EPSG:2100 (Greek Grid 1987) to *targetCRS*=EPSG:4326 (WGS84).

11. Other properties.

- *defaultLang* : This optional property affects the default language tag to be assigned to any string literals created in the output RDF. Unless otherwise specified, the default value is English (i.e., *defaultLang=en*).

5.2.2. Configuration for Reverse Transformation

Reverse Transformation from RDF into geographical files is also controlled via a user-specified configuration file like the one listed in the Annex (Section 9.2.2). Such a file lists several *mandatory* 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), as listed in Table 3.

Input/Output	***CSV specifications***
serialization	delimiter
outputFormat	quote
encoding	***Spatial Reference Systems***
batchSize	sourceCRS
inputFiles	targetCRS
tmpDir	***Other properties***
outputFile	defaultLang
sparqlFile	

Table 3: Properties specified in configuration files for reverse transformation using TripleGeo ver.1.4

More specifically, this configuration file has the following structure and properties:

1. Input and output properties.

- *serialization* is mandatory and concerns the format of the input RDF file(s) and can be set to one of the following values: *RDF/XML*, *RDF/XML-ABBREV*, *N-TRIPLES*, TURTLE (or TTL), and N3.
- *outputFormat*: This mandatory property specifies the format of the output data. Currently, only two formats of *geographical file(s)* are supported, namely *SHAPEFILE*, and *CSV*.
- *encoding*: This optional parameter specifies the encoding (character set) for strings in the *output* data, such as UTF-8, ISO-8859-1, ISO-8859-7, WINDOWS-1253, etc. If not specified, UTF-8 encoding is the considered as default. String literals in *input* RDF triples are always in UTF-8.
- *batchSize*: This optional parameter indicates the number of output records to hold in each batch before storing them to file. This is done for performance, since too frequent writes to disk may slow down the speed of transformation. By default, the current version works with batches of 10 records.

The user must always specify (relative or absolute) *paths* to directories and files used during processing:

- *inputFiles*: The user may specify *multiple* input RDF files (of exactly the same serialization and ontology) separating them by ';' in order to create a single graph with all contents.
- *tmpDir*: The working directory that may be used for storing intermediate files temporarily created during reverse transformation (e.g., the Jena model created on disk from the RDF input).
- *outputFile*: File in which to store the output (reconstructed) records. A single output file will contain all transformed records, even if input comes from multiple files. Files with the same name previously created in the same directory will be overwritten.
- *sparqlFile*: File containing a user-specified `SELECT` query (in SPARQL) that will retrieve results (records) from the input RDF triples. This query should conform with the underlying ontology of the input RDF triples.

In case of CSV output, two additional parameters must be specified (omitted for any other output formats):

- *delimiter*: designates the character delimiting attribute values for each output line (i.e., record). This single character must not appear in any attribute value.
- *quote*: designates the quote character for enclosing string values in attributes.

2. ***Spatial Reference Systems***. If geographic reprojection is required for geometries in the output records, then the following properties must be filled in the configuration:

- *sourceCRS*: the EPSG identifier for the coordinate reference system (CRS) of the input geometries;
- *targetCRS*: the EPSG identifier for the coordinate reference system (CRS) of the output geometries.

In case that *either* of these properties is missing, the respective geometries are assumed to be in the WGS84 reference system (EPSG:4326). TripleGeo works for any valid EPSG reference systems [EPSG] and transforms all geometries in the dataset, e.g., from *sourceCRS*=EPSG:2100 (Greek Grid 1987) to *targetCRS*=EPSG:4326 (WGS84).

3. Other properties.

- *defaultLang* : This optional property affects the default language tag assigned to string literals in the input RDF. Unless otherwise specified, the default value is English (i.e., *defaultLang=en*).

5.3. Execution

As already explained, TripleGeo ver.1.4 supports *two-way* transformation of geospatial features:

- *Transformation* of geospatial datasets from various conventional formats into RDF data. This supports *attribute mappings* into an ontology for RDF features and also specification of a *classification scheme* for assigning categories into input features.
- *Reverse Transformation* of RDF data into de facto geospatial formats (currently, .CSV and ESRI shapefiles). TripleGeo retrieves data from a graph constructed on-the-fly from the RDF data and creates records with a geometry attribute and thematic attributes reflecting the underlying ontology of the input RDF data.

In either case, Java JRE (or SDK) 1.8 (or later) must have been installed. In addition, suitable values must have been set to all required properties in the configuration file, as explained in Section 3.3.2. If triples are to be extracted from a DBMS, then correct credentials must be given in the configuration.

Indicative explanation and usage tips for both transformation modules are given next.

5.3.1. Executing Transformation to RDF

Next, we explain how to use TripleGeo ver.1.4 in order to transform geospatial data into RDF triples:

- In case that triples will be extracted from a geographical file (e.g., ESRI shapefiles) as specified in the user-defined configuration file in `./test/conf/shp_options.conf`, and assuming that binaries are bundled together in */target/triplegeo-1.4-SNAPSHOT.jar*, give a command like this:

```
java -cp ./target/triplegeo-1.4-SNAPSHOT.jar
eu.slipo.athenarc.triplegeo.Extractor ./test/conf/shp_options.conf
```

- If triples will be extracted from a geospatially-enabled DBMS (e.g., PostGIS), a suitable configuration file (e.g., located at `./test/conf/PostGIS_options.conf`) should include all information required to connect and extract data from the DBMS, and the Java command must also invoke a runtime linking to the JDBC driver for enabling connections to the DBMS (e.g., assuming that this JDBC driver is located at `./lib/postgresql-9.4-1206-jdbc4.jar`):

```
java -cp ./lib/postgresql-9.4-1206-jdbc4.jar;./target/triplegeo-1.4-
SNAPSHOT.jar eu.slipo.athenarc.triplegeo.Extractor
./test/conf/PostGIS_options.conf
```

- TripleGeo supports data in *GML* (Geography Markup Language) and *KML* (Keyhole Markup Language). It can also handle *INSPIRE-aligned* GML data for seven Data Themes (Annex I), as well as INSPIRE-aligned XML geospatial metadata. Any such transformation is performed via XSLT, as specified in the respective configuration settings (e.g., `./test/conf/KML_options.conf`) as follows:

```
java -cp ./target/triplegeo-1.4-SNAPSHOT.jar
eu.slipo.athenarc.triplegeo.Extractor ./test/conf/KML_options.conf
```

In all cases, one the process gets finished, the resulting output files can be received in the output directory specified by the user in the configuration.

5.3.2. Executing Reverse Transformation from RDF

This is how to use TripleGeo ver.1.4 in order to transform RDF triples into a geospatial data file:

- In the configuration file, the user must specify one or multiple files that contain the RDF triples that will be given as input to the reverse transformation process.
- The user must also specify a valid SPARQL `SELECT` query that will be applied against the RDF graph and will fetch the resulting records. The path to the file containing this SPARQL command must be specified in the configuration. It is assumed that the user is aware of the underlying ontology of the

RDF graph. If the SPARQL query is not valid, then no or partial results may be retrieved. By default, the names of the variables in the `SELECT` clause will be used as attribute names in the output file.

- In case of ESRI shapefile as output format, make sure that all input RDF geometries are of the same type (i.e., either points or lines or polygons), because shapefiles can only support a single geometry type in a given file.
- Once parameters have been specified in a suitable configuration file (e.g., stored in path `./test/conf/shp_reverse.conf`), the following command can be used to launch the reverse transformation process:

```
java -cp ./target/triplegeo-1.4-SNAPSHOT.jar
eu.slipo.athenarc.triplegeo.ReverseExtractor ./test/conf/shp_reverse.conf
```

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

5.3.3. Known Limitations

Handling large datasets. Judging from our experience with extraction of triples from several geospatial repositories (cf. Section 6.3.1), 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 from each one in multiple concurrent threads. 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) depending on the available system resources.

JDBC connection to geospatial DBMSs. Connection to a geospatial DBMS is performed through a JDBC bridge, so a suitable driver should be available. Until ver.1.1, TripleGeo was shipped with several such freely available drivers (e.g., for PostgreSQL), although certain software vendors restrain usage of such tools only to customers that have purchased their DBMS platform (e.g., IBM DB2). Starting from ver.1.2, TripleGeo *no longer includes any JDBC drivers*, so users may either specify them in the *pom.xml* and rebuild the application or directly invoke a particular .jar that contains the necessary drivers at the command line. So, before attempting to execute TripleGeo against data residing in any DBMS, the user should make sure that the necessary JDBC driver(s) for that version of the DBMS software are available in their system and accessible by the TripleGeo utility.

Interacting with Oracle 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 relates to 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 6.3.1.

Besides, JDBC connections to Oracle DBMS use some "random numbers" to encrypt the connection information, and the lack of these numbers may cause failures. To solve this issue, it is necessary to define a different origin for random numbers than the default. The parameter to define the origin of random numbers is: `-Djava.security.egd=`, and one of the available choices inn Linux for this "random origin" is `/dev/urandom`. Hence, invoking TripleGeo for extracting and transformation geospatial data stored in an Oracle database should include the following directive in the Java command line so as to avoid serious delays in JDBC connections:

```
-Djava.security.egd=file:/dev/urandom
```

5.3.4. Demonstration

In this Section, we walkthrough indicative executions of TripleGeo by making use of the configuration settings (Section 9.2), attribute mappings (Section 9.3) and classification schemes (Section 9.4) listed in the Annex regarding a sample POI dataset. Note that TripleGeo is a command-line tool without a graphical interface of its own. However, once it is integrated in the SLIPO Workbench, it will be possible to configure its parameters and launch execution (either transformation to RDF or reverse transformation from RDF) in a user-friendly fashion.

5.3.4.1. Example for Transformation to RDF

In this example regarding *transformation* to RDF of a shapefile dataset (containing a sample of one million POIs extracted from OSM) according to a configuration file (`shp_options.conf`), the user should give the following command to start the transformation process with TripleGeo ver.1.4:

```
java -cp ./target/triplegeo-1.4-SNAPSHOT.jar
eu.slipo.athenarc.triplegeo.Extractor ./test/conf/shp_options.conf
```

If all paths are properly defined and the configuration is valid, the transformation process is launched successfully and the user is notified accordingly:

```
Encoding: UTF-8

Conversion mode: STREAM

Output serialization: N-TRIPLES

Transformation will take place from EPSG:2100 to EPSG:4326 reference system.
```

In case that a *classification* scheme has been specified (like the sample ones listed in Section 9.4), then TripleGeo proceeds to transform its contents into RDF and keeps in memory their corresponding URIs in order to create links to them from POIs in a subsequent step. The following notifications correspond to transformation of categories:

```
Classification hierarchy reconstructed from CSV file.

2018-03-12 12:04:16.727 GMT Thread main completed successfully in 216 ms. 181 records transformed into 709 triples and exported to N-TRIPLES file: ./test/output/poi_classification.nt.

2018-03-12 12:04:16.899 GMT Classification hierarchy read successfully!
```

Afterwards, TripleGeo starts consuming input POI data and creating RDF triples according to the chosen transformation mode. In case that multiple threads are employed, each one periodically (every 1000 input records) notifies the user on its progress:

```
2018-03-12 12:04:21.573 GMT pool-1-thread-1 Processed 24000 records...
```

Once a thread concludes transformation of its input data, it issues the following message:

```
2018-03-12 12:05:36.448 GMT Thread pool-1-thread-1 completed successfully in 77250 ms. 1000000 records transformed into 23510309 triples and exported to N-TRIPLES file: ./test/output/points.nt.
```

At the end of the process, TripleGeo notifies about the spatial extent of the transformed geometries (MBRS in WGS84), as well as the path to all output files containing RDF triples. Note that multiple such files may be obtained in case that multiple threads have been employed to handle disjoint pieces of the input POI data:

```
MBR of transformed geometries: X_min=-31.2638012, Y_min=29.9851487, X_max=46.6462048, Y_max=80.5156505

2018-03-12 12:05:36.451 GMT Transformation process concluded successfully in 79548 ms.

RDF results written into the following output files: [./test/output/poi_classification.nt, ./test/output/points.nt]
```

5.3.4.2. Example for Reverse Transformation

In this example, *reverse transformation* from RDF datasets to ESRI shapefile format is specified according to a configuration file (`shp_reverse.conf`) and it can be invoked as follows:

```
java -cp ./target/triplegeo-1.4-SNAPSHOT.jar eu.slipo.athenarc.triplegeo.ReverseExtractor ./test/conf/shp_reverse.conf
```

If all paths are properly defined and the configuration is valid, the reverse transformation process is launched successfully and the user is notified accordingly:

```
Encoding: UTF-8
```

```
Transformation will take place from EPSG:4326 to EPSG:2100 reference system.
```

Afterwards, TripleGeo starts creating a disk-based RDF model that will include all triples read from the RDF files specified as input. The tool provides a notification once each piece of input RDF data is loaded to the RDF graph:

```
2018-03-16 14:54:45.515 GMT Initializing RDF graph to hold input triples...
Done!

2018-03-16 14:54:46.550 GMT Reading triples from file
./test/output/poi_classification.nt... Done!

2018-03-16 14:54:46.704 GMT Reading triples from file
./test/output/points.nt... Done!
```

Once all RDF input is loaded, TripleGeo issues notifications about the size of the created graph, as well as specific warnings in case that restrictions apply regarding geometry types (e.g., ESRI shapefiles can only support a single geometry type):

```
2018-03-16 15:00:33.092 GMT RDF graph loaded successfully and contains
14510626 statements in total.

All geometries are expected to be of type POINT in order to be included in
this shapefile.
```

Afterwards, TripleGeo applies the user-specified SPARQL query against this RDF graph and starts emitting output records notifying the user about its progress:

```
2018-03-16 15:00:48.710 GMT Processed 42400 records...
```

Once all records are reconstructed, TripleGeo notifies the user and points to the file where the output is available:

```
2018-03-16 16:09:03.005 GMT 1000000 results retrieved from the RDF graph.
1000000 features created.

2018-03-16 16:09:03.432 GMT Reverse transformation process terminated
successfully!

Records were written into output file:./test/output/points_reconstructed.shp
```

6. Experimental Evaluation

In this Section, we examine the efficiency and scalability of the TripleGeo software, as it has evolved until the current stage of the project. We report results from a comprehensive validation of TripleGeo against large POI datasets extracted from OpenStreetMap [OSM] and stored in various geospatial repositories.

In Section 6.1, we describe the original OSM datasets and present some of their subtle characteristics, with particular emphasis on their classification into categories, the variety of their geometric representations, as well as the multitude of thematic attributes that may be extracted. In Section 6.2, we discuss the experimental setup, explaining the performance metrics assessed in our tests. Finally, in Section 6.3, we discuss performance results regarding both transformation to RDF and reverse transformation from RDF, confirming that TripleGeo can now handle any number of thematic attributes and map them to a given POI ontology. Most importantly, these results testify that TripleGeo ver.1.4 has already achieved *orders of magnitude performance gains* compared to its original release, and can now *efficiently transform millions of POIs in a few minutes* even without any sophisticated data partitioning schemes, thus paving the way for even more advanced scalability in its forthcoming releases.

We have also successfully tested SLIPO against POI data provided by the SLIPO industrial partners for the various use cases. This transformation process also required preparation of custom mappings per dataset and application of various classification schemes. Table 8 offers a preliminary overview of the original POI datasets and their resulting RDF representations; a detailed description of POI characteristics, their applicable mappings to the SLIPO ontology, as well as their diverse classification schemes will be included in the Confidential Deliverable D5.1 "Specification of the Pilots".

Provider	Dataset	Spatial coverage	Input records	Output triples
WiGeoGIS	Herold	Austria	350,053	26,214,671
WiGeoGIS	Wigeoapi (OSM)	Austria	156,594	3,366,026
TomTom	MultiNet	Austria	311,078	13,587,632
TomTom	MultiNet	Germany	2,425,708	106,650,699
TomTom	MultiNet	Greece	127,910	4,998,524
GET	get-pois_v07	Greece	72,373	3,313,133
Athena RC	OpenStreetMap	Europe	7,447,693	121,439,699

Table 4: Datasets provided by SLIPO partners and transformed into RDF using TripleGeo ver.1.4

6.1. Datasets

In order to verify the transformation capabilities of TripleGeo, we have extracted POI data concerning all Europe from the OpenStreetMap (OSM) database [OSM], available under the Open Database License (ODbL). In particular, geospatial and thematic information in OSM is distinguished in the following core elements:

- *Nodes*. These are points with a geographic position, stored as coordinate pairs (longitude, latitude) georeferenced in WGS84. They are used to represent any kind of points, from road intersections to points of interest, as well as vertices of more complex spatial entities (lines, polygons, etc.).
- *Ways*. These are ordered lists of nodes, representing a polyline or a polygon. They are used for representing both linear features (e.g., streets, rivers), and areas (e.g., parks, lakes).
- *Relations*. These are ordered lists of nodes, ways and relations. They are used for representing the relationship of existing nodes and ways.
- *Tags*. These are key-value pairs (both arbitrary strings). Even though an ontology [OSMOnto] is recommended for such tags, this is not strictly followed, so free tags may be used in representing metadata about the map objects (such as their type, their name and their physical properties).

Obviously, the data schema and format in OSM is not specifically tailored to representing and describing POIs, but generally any type of map elements. In order to be able to obtain POIs from OSM, we had to extract: (a) the *detailed geometry* of each POI and (b) the complete set of *tags* associated with an OSM feature. Note that the geometry can be more complex than a centroid; for example, in OSM an archaeological site may be represented by its boundary, and a popular scenic route by a polyline (linestring). We have made use of the OGR/GDAL software [GDAL], which can extract all features contained within an OSM file, including their detailed geometry (stored in an attribute called *shape*), as well as the full set of tags (*all_tags*) available for each feature, i.e., a list of key-value pairs in *hstore* representation [HSTORE] for all user-specified OSM tags identified in a given feature. We have extracted such features within the spatial extent of Europe from the OSM database.

Category	Type
POI	HAMLET, VILLAGE, PEAK, CRANE, SUBURB, TOWN, CAVE, TOWERLOOKOUT, BUNKER, EMBASSY, MINE, TOWERCOMMUNICATION, CITY, PEAK1
TRANSPORT	BUSSTOP, FUEL, STATION, TRAMSTOP, RENTALCAR, MARINA, AIRPORT, SUBWAY, LIGHTHOUSE, TERMINAL
SHOP	DEPARTMENTSTORE, SUPERMARKET, CONVENIENCE, CLOTHES, HAIRDRESSER, BAKERY, CARREPAIR, DIY, CAR, FLORIST, BICYCLE, BUTCHER, SHOES, ALCOHOL, HIFI, KIOSK, JEWELRY, MOTORCYCLE, BOOK, PHONE, LAUNDRETTE, CONFECTIONERY, COPYSHOP, GIFT, VENDINGMASCHINE, MARKETPLACE, TOYS, COMPUTER, GARDENCENTRE, GREENGROCER, PET, NEWSPAPER, TOBACCO, FISH, HEARINGAIDS, MUSIC, VIDEORENTAL
FOOD	RESTAURANT, CAFE, FASTFOOD, PUB, BAR, ICECREAM, BIERGARTEN
TOURIST	MEMORIAL, ATTRACTION, FOUNTAIN, THEATRE, ART, MUSEUM, ARCHAELOGICAL, CASTLE2, BEACH, RUINS, MONUMENT, NIGHTCLUB, CINEMA, INFORMATION, WINDMILL, ZOO, THEMEPARK, CASTLE, BATTLEFIELD, WRECK
EDUCATION	SCHOOL, NURSERY, UNIVERSITY, COLLEGE
LANDUSE	GRASS, CONIFEROUSDECIDUOUS, ALLOTMENTS, SWAMP, QUARY, MILITARY, SCRUB, HILLS, DECIDUOUS, CONIFEROUS
AMENITY	PUBLICBUILDING, POSTOFFICE, TOWNHALL, FIRESTATION, LIBRARY, PLAYGROUND, POLICE, COURT, PRISON
POW	CHRISTIAN, ISLAMIC, UNKOWN, JEWISH, BUDDHIST, HINDU, SIKH, BAHAI, JAIN, SHINTO
ACCOMMODATION	HOTEL, CAMPING, CHALET, HOSTEL, ALPINEHUT, CARAVAN, MOTEL
HEALTH	PHARMACY, DOCTORS, HOSPITAL, DENTIST, VETERINARY, HOSPITALEMERGENCY

Category	Type
SPORT	LEISURECENTER, SOCCER, STADIUM, SWIMMING, TENNIS, GOLF, HORSE, CLIMBING, BASKETBALL, SHOOTING, BOWLING, GYM, SNOOKER, GYMNASIUM, SKATING, CRICKET, MOTORRACING, BASEBALL, CANOE, MINIATURGOLF, FOOTBALL, SKIINGDOWNHILL, ICESKATING, ARCHERY, WATERSKI, SURFING, DIVING
MONEY	BANK, EXCHANGE
BARRIER	BLOCKS
WATER	WEIR, TOWER, DAM

Table 5: Classification scheme applied against POIs extracted from OpenStreetMap

The result of such extraction may refer not only to POIs, but also to other OSM features (e.g., linestrings representing road segments). What is lacking from OGR/GDAL is the ability to identify POIs and classify them into specific *categories*. In order to extract POIs from OSM, we have configured a filter for OGR/GDAL that can extract OSM features into five layers stored into respective tables in a PostgreSQL/PostGIS database:

- *points* : OSM node elements that have significant tags attached;
- *lines* : OSM way elements that are recognized as non-area;
- *multilinestrings*: OSM relation elements that form a multilinestring (i.e., `type = 'multilinestring' OR type = 'route'`).
- *multipolygons* : OSM relation elements that form a multipolygon (i.e., `type = 'multipolygon' OR type = 'boundary'`), and "way" features that are recognized as area; and
- *other_relations* : OSM relation elements that do not belong to the above two layers.

Then, based on the tags available for each OSM element, we categorized the resulting records according to the two-tier classification scheme depicted in Table 5. This classification was obtained from OsmPoisPbf [OsmPoisPbf] and defines a user-specified filter file that maps a specific combination of OSM tags into a single class. This scheme consists of a *primary category* (one of the 15 categories listed in Table 5; each feature has exactly one category from this list) and a *secondary type* selected among 167 distinct values identified in OSM tags (each feature has exactly one type selected from the list). By parsing the tags available in each element (listed in the types column in Table 5), we determined their corresponding category (the first column in Table 5).

Finally, we compiled into a single table in the PostgreSQL/PostGIS database all OSM features classified as POIs with the aforementioned process. This table contains 7,447,693 POI records over Europe as illustrated in Figure 11; in the sequel, we refer to this dataset as *OSM 7.4M POIs*. In addition to the basic attributes extracted with OGR/GDAL, we specified filters with SQL queries over the set of OSM tags in order to isolate particular attributes concerning POIs. As listed in Table 6, these extra attributes refer to *address* and *contact* information, as well as other complementary attributes. Note that much more information from original OSM tags can also refer to POIs (e.g., payment methods, access to services, etc.), which we plan to extract after assessing their potential utilization in the POI data integration lifecycle for the particular pilot cases.

Basic attributes	Address-related attributes	Contact-related attributes	Other attributes
geometry	street name	phone number	international name
OSM identifier	house number	fax number	category
name	postal code	email address	country
type	city	webpage	opening hours

Table 6: Attributes extracted from OpenStreetMap concerning POIs

Note that geometric information for these POIs concerns not only *Points* (in longitude/latitude coordinates) according to the OGC geometry types, but also *LineStrings*, *MultiLineStrings*, *MultiPolygons*, as well as Geometry Collections, as listed in Table 7. Typically for OSM data, all geometries in this dataset are georeferenced in WGS84 (EPSG:4326).

OGC Geometry type	OSM 7.4M POIs	OSM 1M POIs
Point	5420386	738877
LineString	16361	2718
MultiLineString	3	0
MultiPolygon	2005337	257711
GeometryCollection	5606	694
TOTAL	7447693	1000000

Table 7: Number of POIs available per geometry type in the two OSM datasets

In order to conduct tests that respect the capabilities of earlier versions of TripleGeo, we have also taken a subset of this OSM dataset by randomly selecting one million records. This dataset, called *OSM 1M POIs*, is used to compare performance in transformation amongst subsequent versions of TripleGeo, in handling an increasing number of attributes, as well as in testing the reverse transformation functionality.

In either dataset, data was used "as is" without modifying original geometric or thematic information. Since these datasets reside in a *PostgreSQL/PostGIS* database, we have extracted them in three other formats:

- *CSV* (comma separated values), where attribute values are actually separated by '|';
- *ESRI shapefile*, which is a de facto format for geographical vector information; and
- *Oracle Spatial*, which handles geometries in its own custom representation (`SDO_GEOMETRY`).

These four alternative formats where employed in order to verify the correct functionality of TripleGeo when data has to be retrieved from different spatial repositories.

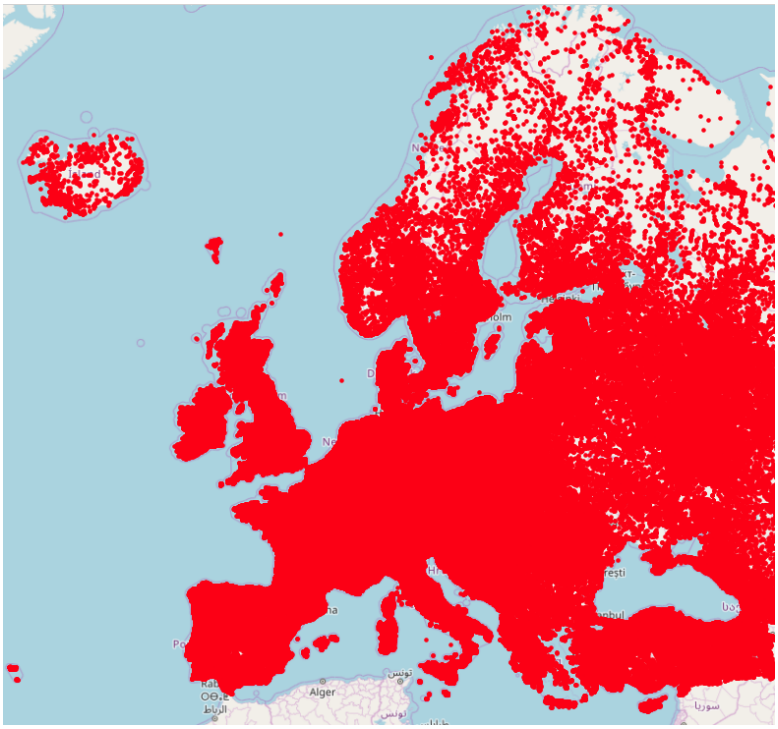


Figure 11: Points of Interest (POI) extracted from OpenStreetMap across Europe

6.2. Experimental Setup

We have repeatedly applied TripleGeo against various sample and third-party datasets on both MS Windows and Linux environments in order to verify its correct functionality and smooth operation.

In terms of validating *performance* and *scalability* of TripleGeo against POI data, we have specifically carried out tests against the two OSM datasets presented in Section 6.1. These experiments were conducted on a Virtual Machine running Debian (Linux 3.16.0) on an Intel Core i7-3820 CPU with 4MB cache at 2.2 GHz. This VM was given 8GB RAM, 1GB swap, 4 (virtual) CPU cores and 300GB disk of storage space. All datasets were locally available in the same VM where TripleGeo is running, so no network delays were involved.

Since all datasets are available in either structured geographical files or geospatially-aware DBMSs, the XSLT mode in TripleGeo is not involved, as it can be only applied on semi-structured data (GML, KML, XML). Hence, we only present results concerning the GRAPH, STREAM, and RML modes.

Each experiment was executed in *cold runs*, i.e., invoking TripleGeo immediately after all caches of the operating system are cleared, the DBMS is re-started (if used), and no data is loaded into the system's main memory.

In the experiments, we primarily measure the *clock time* (in seconds) required to transform a given dataset. This refers to the *end-to-end* time elapsed since the beginning of the process, i.e., including accessing the repository (i.e., the file or DBMS holding the input data), applying mappings and classifications, transforming all input features, and also writing the resulting RDF triples into files on disk. Note that computations in TripleGeo are both I/O and CPU intensive. In particular, heavy I/O is involved when reading large input datasets and finally writing the output triples; note that, in general, RDF triples are inherently verbose when

compared with original records. Besides, most CPU activity is dedicated into actual transformation, when mappings are applied to each input feature with possible coordinate reprojections.

In the two transformation modes of TripleGeo that work in a *streaming* fashion (i.e., *STREAM* and *RML*), we also provide results regarding the *average throughput*, i.e., the rate in triples/sec at which TripleGeo generates RDF triples as it progressively consumes the input dataset.

6.3. Performance Results

Next, we provide indicative results for a set of experiments validating performance and scalability of TripleGeo both in *transformation* from geospatial formats to RDF and in *reverse transformation* from RDF back to geospatial files.

6.3.1. Performance of Transformation to RDF

6.3.1.1. Performance across Successive Software Releases

The first experiment compares performance of TripleGeo across its successive software releases, starting from ver.1.1 available in the beginning of the project until current ver.1.4. The plot in Figure 12 illustrates the time it takes in these four releases of TripleGeo to transform the *OSM 1M POIs* dataset, i.e., one million POIs in Europe available in a PostgreSQL/PostGIS repository. In order to provide a fair comparison regarding performance amongst versions, we only examine *four basic attributes* in each POI (first column in Table 6). This restriction is imposed by ver.1.1, because this release (inherited from the GeoKnow project and used as a starting point in our development) could only support up to four attributes; by default, one of these attributes is the geometry and the rest are thematic ones. From the plot, it is obvious that the STREAM mode is extremely fast, almost an order of magnitude faster than the GRAPH mode originally available in ver.1.1. Note that RML is slower than STREAM because it must apply each one of the specified performers in the mapping against each input record, even in case of null values in the respective attributes. However, RML is faster than GRAPH, as it also works in a streaming fashion. Indeed, in either STREAM or RML each input record is transformed on-the-fly and readily propagated to the output, without the cost of building and updating a disk-based RDF model as in the GRAPH mode. However, even for the GRAPH mode, in ver.1.4 we have achieved to drop the cost by almost 50% by streamlining intermediate computations per record while also avoiding creation of a temporary set of triples before writing the results to the output file. Finally, notice the small increase in the cost for ver.1.3 compared with its predecessor ver.1.2. Indeed, ver.1.3 was the first release that included (partial) support for classification schemes, and practically emitted extra triples with string literals for the identified category. The current release (ver.1.4) maintains the user-specified classification scheme in memory; when transforming a record, it actually creates a link to the URI of its respective category. Not only is this conforming to the SLIPO ontology for POIs, but evidently it is more efficient in practice across all transformation modes.

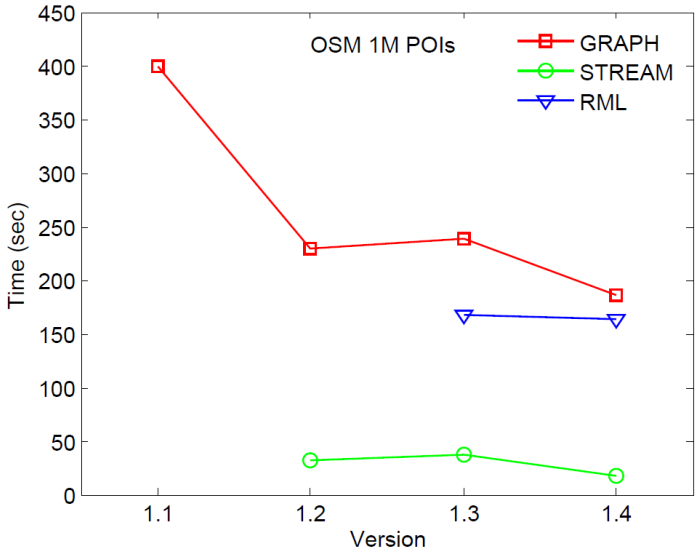


Figure 12: Performance of the various transformation modes as supported by each successive release of TripleGeo when dealing with *four attributes* per record

6.3.1.2. Performance with Increasing Number of Attributes

This experiment examines performance of TripleGeo ver.1.4 when dealing with increasing number of attributes per input record. Figure 13 depicts the overall execution time of transforming one million POIs (*OSM 1M POIs*) to RDF, when each POI includes a varying number of attributes as listed in Table 6. Observe that the scale along time measurements is logarithmic. Clearly, the more the attributes available per input record, the more the resulting RDF triples; hence, the cost should increase linearly with the number of attributes. This is exactly the case with the RML mode, since the RML performers employed in transformation must examine each attribute value irrespective of NULL values. But notice that cost for STREAM and GRAPH modes seem practically unaffected when dealing with 12 attributes instead of 8, i.e., when examined contact information per POI as well. This is because the majority of POIs in that dataset actually lacks most of this information and those extra four attributes have NULL values. Hence, YAML mappings are not applied at all in such cases and thus save processing cost. Of course, STREAM mode is at least an order of magnitude faster from GRAPH, and even more faster than RML for any number of attributes per input record. Even when all 16 attributes per record need be transformed, the process in STREAM mode takes less than 65 seconds to conclude, as opposed to 587 seconds in GRAPH and 964 seconds in RML mode. The only case where RML fares better than GRAPH is when only four (basic) attributes are examined. In that case, the cost for creating the disk-based model and then extracting the RDF triples is slightly greater (22.5 seconds higher) than applying the series of RML performers and collecting the triples for all input records. Overall, it is evident that TripleGeo can efficiently handle a varying number of thematic attributes per record with linear (worst-case) or sublinear (amortized) increase in transformation cost.

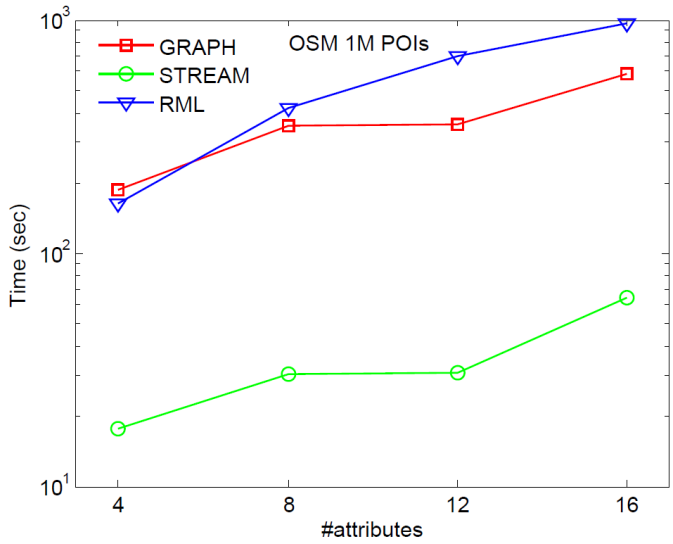


Figure 13: Performance of the various transformation modes in TripleGeo ver.1.4 when handling an increasing number of attributes per input record

6.3.1.3. Scalability with Increasing Data Volumes

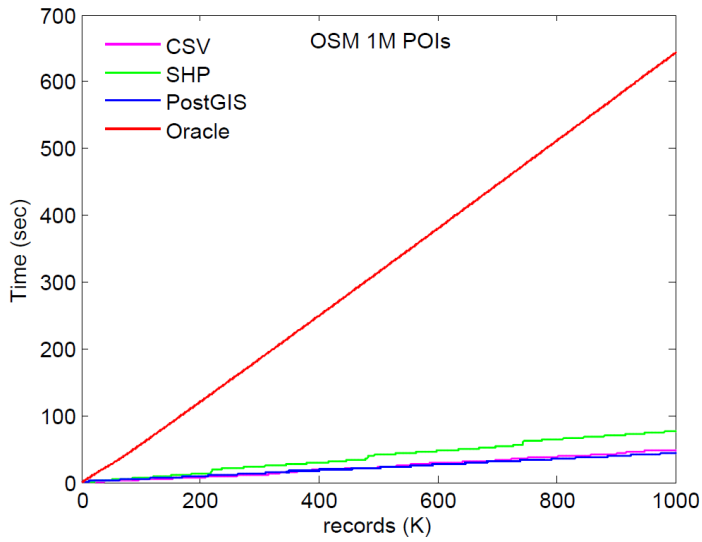


Figure 14: Performance of TripleGeo ver.1.4 (in STREAM transformation mode) when handling an increasing number of input records from different data repositories

The next experiment examines scalability of TripleGeo against increasing volumes of input records, up to one million POIs available across Europe (dataset: *OSM 1M POIs*). In this test, we employ TripleGeo in its STREAM mode, as this is manifestly the most efficient. We measure execution times when accessing the same input data (all 16 attributes in Table 6) available from different repositories: either de facto file formats (CSV, ESRI shapefile) or a geospatially-aware DBMS (PostGIS, Oracle Spatial). From the plot in Figure 14, it is evident that performance grows linearly with the amount of input records retrieved from each repository. This is expectable, since the STREAM mode handles each record in isolation and promptly emits the resulting triples before proceeding to handle the next record in the data. However, there is some divergence in performance amongst repositories. Indeed, when input data comes from CSV or PostGIS, transformation to RDF proceeds rapidly. This should be expected for a CSV file, as this is a plain text format and lines in the file can be consumed swiftly and then turned into records. But, the JDBC driver for PostGIS seems to be almost

equally fast in providing records from the result set obtained after the SQL query is applied against the DBMS. In contrast, the respective JDBC for Oracle provides records at a much slower pace, hence the significant slowdown in performance. When input is available in ESRI shapefile format (SHP), then performance is also slower than CSV, because each record must first be parsed by the GeoTools library. This parsing almost doubles the cost w.r.t. CSV, before applying any transformation to RDF, hence the increase in the total execution time. Overall, this experiment indicates that the type of data repository plays an important role in the rate at which input data is accessed and has a strong impact in transformation cost.

6.3.1.4. Performance with Multiple Execution Threads

The next set of experiments are applied against the much larger dataset of OSM 7.4M POIs with the objective to examine performance of TripleGeo ver.1.4 when multiple execution threads are employed in transformation. More specifically, the input dataset is split into several (up to 16) equal parts in CSV format, and a separate thread applies transformation to RDF under the same mode (GRAPH, STREAM, or RML).

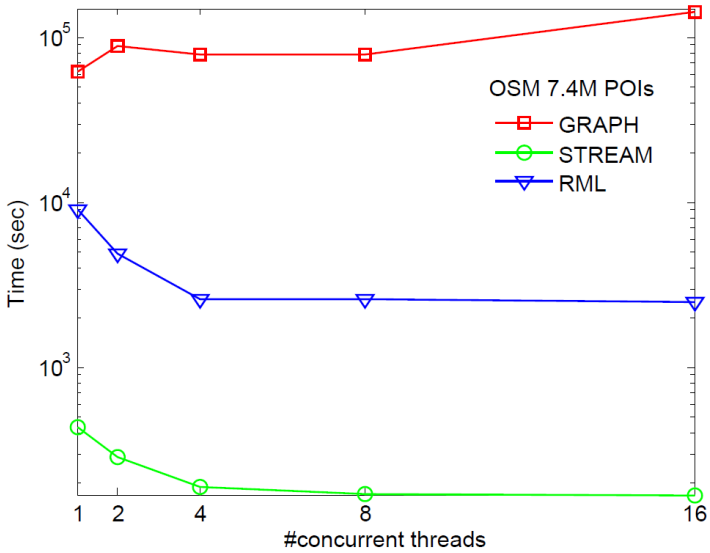


Figure 15: Performance of the various transformation modes in TripleGeo ver.1.4 when employing a varying number of concurrent threads

Figure 15 illustrates the total execution time (shown in logarithmic scale) for transforming the entire dataset with a varying number of *concurrent* threads. Obviously, when a single thread is used, each mode deals with the entire dataset. In this case, we can observe that STREAM is an order of magnitude faster than RML (which also works in streaming fashion, but has to apply a series of RML performers against each record), and almost two orders of magnitude faster than GRAPH. The same pattern persists when multiple concurrent threads are used in each mode. It is no wonder that transformation cost drops with extra threads for STREAM and RML modes, as each thread handles separately a smaller chunk of the data and takes out the most of available system resources (memory, CPU). Of course, this performance gain gets less pronounced when invoking more than 8 threads, as the system cannot resourcefully sustain all of them concurrently and context switching inevitably ensues.

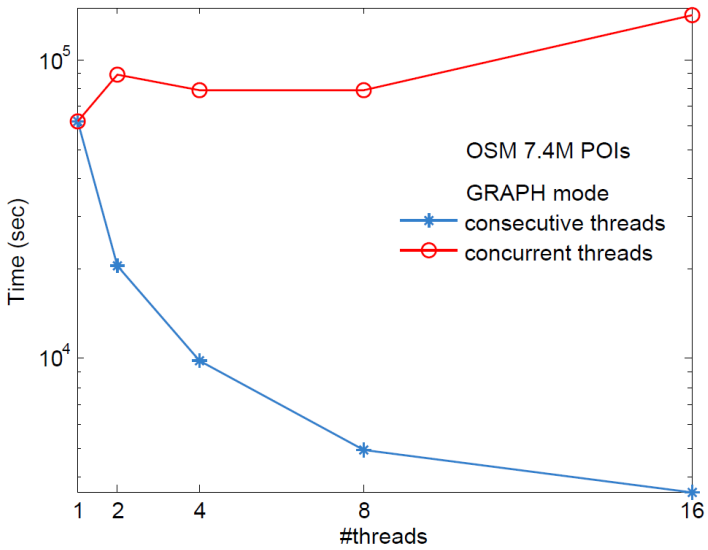


Figure 16: Performance of the GRAPH transformation mode in TripleGeo ver.1.4 when using multiple, either consecutive or concurrent threads to process input

However, the cost for the GRAPH mode increases when multiple *concurrent* threads are involved as the plot in Figure 15 testifies. This is primarily due to the high I/O interaction of this mode, as each thread has to maintain its own disk-resident RDF model. In addition, RDF indices have to be constantly updated upon additions of new statements to a model, incurring extra CPU and I/O cost. The available memory is exhausted once these models grow bigger and the system cannot efficiently maintain all of them concurrently. To verify this effect, we repeated the same test specifically in GRAPH mode when each thread is not invoked concurrently with the rest, but sequentially. More specifically, once a thread concludes transformation of its own part of the data, another one is invoked to handle another part. So, the same number of *consecutive* threads is applied, each one running alone without any impact from other concurrent threads of TripleGeo. The plot in Figure 16 confirms that using more threads against smaller chunks of data can boost performance even in GRAPH mode, provided that these threads are invoked in a sequence and not concurrently. Indicatively, transformation of the entire dataset with 16 *consecutive* threads concludes in less than 60 minutes, but when the same number of threads are invoked *concurrently* execution time soars to almost 24 hours (mind the log scale across time measurements)! Given sufficient system resources (CPU, memory), the smaller the input data chunk, the more efficient its loading to the disk-based model and the sooner its transformation to RDF. In contrast, executing those threads in parallel is counter-productive as it soon exhausts available resources and imposes frequent context switching. Even when executed with multiple consecutive threads, the GRAPH mode still trails behind the other two transformation modes (STREAM, RML) that run in a streaming fashion. This is inevitable, as these latter methods work entirely in main memory and only touch the disk for storing the resulting RDF triples; instead the GRAPH mode entails heavy interaction with the disk and this is the cause of delays. Of course, Jena allows memory-resident models to be used, but this cannot be applied even against input datasets of moderate size (less than a million records with our VM capabilities).

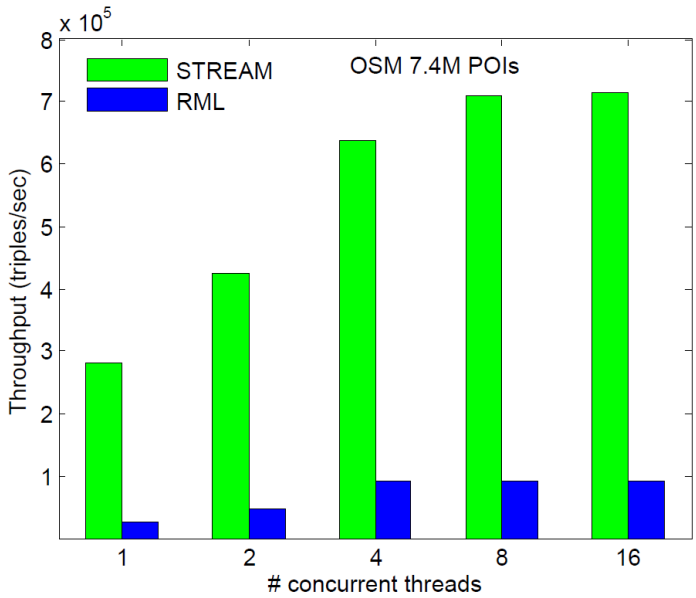


Figure 17: Throughput of the streaming transformation modes in TripleGeo ver.1.4 when using multiple concurrent threads to process the input

Regarding especially the two streaming modes (STREAM, RML), Figure 17 depicts the average throughput, i.e., the rate (in triples/sec) at which RDF triples are generated with a varying number of concurrent threads. As expected, in both transformation modes throughput is increasing with extra threads, but the effect diminishes when reaching the ceiling of available system resources (i.e., when more than 8 threads are used). Again, the STREAM mode can generate up to 715,000 triples/sec and is much more efficient than the RML mode, as the former readily applies YAML mappings tailored for each attribute, whereas the latter has to iterate over all RML performers in order to identify the one with the RML mapping suitable for the given attribute value. It should be also noted that RML mode generates extra triples for RDF properties specified in the RML mappings (e.g., characterization of the name of a POI as official or brand name) even in presence of NULL values in the respective attributes. Although this has no implications to the correctness of the RDF output, it inflates its size with superfluous triples. In the future, we will attempt to alter the way RML performers handle such cases in order to avoid generation of such triples.

6.3.2. Performance of Reverse Transformation

Remember that the reverse transformation functionality of TripleGeo ver.1.4 aims to reconstruct geospatial entities as records with thematic attributes from RDF datasets. Although performance of this module is not critical in the context of the SLIPO project because reverse transformation is not involved in the POI lifecycle, it is still important when the results of POI data integration (i.e., after interlinking, fusion, and enrichment) should be returned back to users in a de facto geospatial format. Of course, there exists an impedance mismatch in this reverse direction, given that the SLIPO ontology is semantically more expressive than the conventional POI schemata, thus POI attributes, relations and metadata will be richer than what can be supported by conventional formats. Presently, we apply reverse transformation to CSV or ESRI shapefile over the results received after transformation to RDF of the OSM 1M POIs dataset. As mentioned in Section 4.1, reverse transformation requires a SPARQL query against the RDF dataset, so we specify this query in order to reconstruct exactly the *same attributes* available in the original data. Apart from

testing the capabilities of the reverse transformation module, this experiment can also verify that no information is lost when the original dataset is transformed into RDF by TripleGeo.

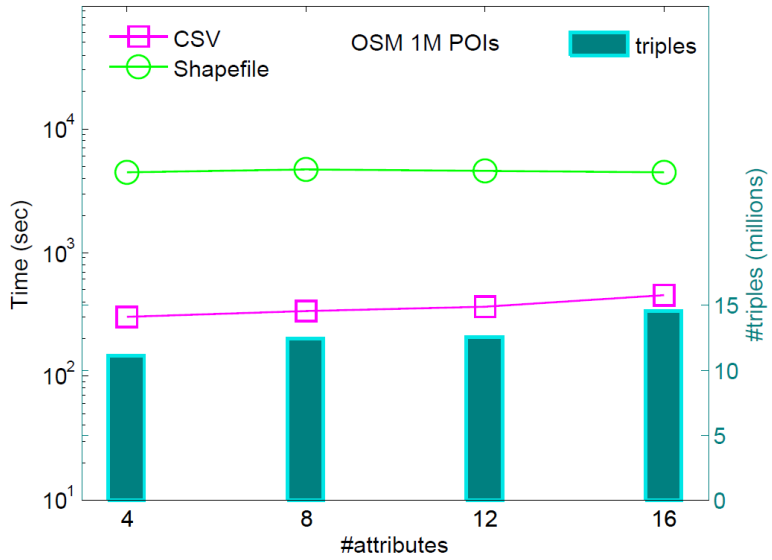


Figure 18: Performance of *reverse transformation* in TripleGeo ver.1.4 for varying number of attributes per record

Figure 18 shows the time (in logarithmic scale along the left *y*-axis) taken by reverse transformation to reconstruct the data in CSV and ESRI shapefile formats when a varying number of original attributes had been extracted into RDF (as listed in Table 6). Note that, with an increasing number of extra attributes more RDF triples had been generated, as indicated with the bar plots. So, in reverse transformation, it takes more time to restore these triples into a disk-based RDF model and subsequently extract records by linking all available properties for each entity in the model. This is more evident when the result of reverse transformation is written into plain CSV format, which concludes very quickly. Even for 16 attributes, the restored CSV dataset is ready in almost 7.5 minutes, even though it has to get this information from more than 14.5 million triples. However, reverse transformation to shapefiles incur an order to magnitude more cost, up to 74 minutes for 16 attributes in this test. This is because the GeoTools library must first create a feature record and assign the respective attribute values (including the geometry) before storing this record into the shapefile. In addition, GeoTools performs tests regarding validity of geometries and their conformance to the geometry type, since shapefiles support only a specific type of geometries (i.e., either points or polylines or polygons, but not a mixture of them). Although such tests seem to incur a certain delay in the construction of shapefiles, this sometimes may be preferable as it guarantees consistency in the geospatial information that should not always taken for granted with WKT representations in CSV files. Finally, it should be noted that we have compared the reconstructed CSV files and shapefiles with the original data, confirming that we received the same number (one million) of records with no extra NULL values in any attribute. Overall, this experiment confirms that the reverse transformation functionality of TripleGeo can reconstruct original data with no loss of information.

7. Conclusions

In this Deliverable, we presented the mapping and transformation service for POIs as implemented by M15 of the project. In particular, we discussed the progress regarding TripleGeo, the *entry point* for POI datasets in the SLIPO lifecycle.

TripleGeo can take as input not only a variety of de facto geographical files (*e.g., ESRI shapefiles*) and semi-structured formats (*e.g., GML, GeoJSON*), but may also access spatial tables hosted in renowned geospatially-aware DBMSs (*e.g., Oracle Spatial, PostGIS*). Further, it copes with most common geometry data types, including more complex geometries, as well as on-the-fly transformations into other coordinate reference systems. Geometries can be exported in several serialized RDF formats, and the resulting triples can be generated according to user-specified ontologies and vocabularies. Apart from geometries, TripleGeo enables the extraction of all thematic (*i.e., non-spatial*) attributes available in the input data, as well as the definition of classification schemes for assigning categories in the POI features. In a comprehensive evaluation study against real-world datasets containing millions of POIs across Europe, we confirmed that TripleGeo can efficiently access data from various repositories, can handle varying geometry representations and multiple thematic attributes applying user-specified attribute mappings, and can also comply with classification schemes that assign categories to POI features. Most importantly, this software can transform millions of POIs in a few minutes, confirming its robustness and versatility of the software and testifying its potential for efficient handling of even larger POI datasets.

In the future, we plan even more extensions and improvements on the SLIPO transformation service. First, accessing even more de facto POI formats and possibly POI data available through OGC-compliant WFS APIs would be also a useful addition to the current functionality. Regarding mappings, we will examine the possibility of *semi-automatic workflows* for guiding the user into creating new mappings (*e.g., in transforming datasets whose schema is not mapped to an existing POI ontology*). In terms of applying attribute mappings, and given that the STREAM transformation mode seems the most promising in terms of performance, we plan to modify it in order to accept a *unified specification of mappings in RML* as an alternative to the currently custom YAML mappings; although the latter are much easier for users to specify, they lack expressiveness and cannot handle complex ontologies. In terms of *reverse transformation* from RDF into de facto geospatial formats, we will search for schemes that allow incorporation of the maximum amount of semantic (linked, enriched, fused) POI information and metadata into the returned attributes of conventional POI formats. However, the most important challenge regarding transformation is *scalability* with increasing data volumes. We will focus on effective data partitioning schemes that can be applied across a *parallelized* framework in cluster infrastructures for transforming massive collections of POIs and generating the resulting RDF triples with minimal latency.

8. References

[AllegroGraph] Franz Inc. AllegroGraph Semantic Graph Database. https://franz.com/agraph/allegrograph/

[ADL+09] S. Auer, S. Dietzold, J. Lehmann, S. Hellmann, and D. Aumueller. Triplify: Light-weight linked data publication from relational databases. In Proceedings of the 18th International Conference on World Wide Web (WWW 2009), pp. 621-630, Madrid, Spain, April 2009.

[BK16] K. Bereta and M. Koubarakis. Ontop of Geospatial Databases. In Proceedings of the 15th International Semantic Web Conference (ISWC), pp. 37-52, Kobe, Japan, October 2016.

[BXK+16] K. Bereta, G. Xiao, M. Koubarakis, M. Hodrius, C. Bielski, and G. Zeug. Ontop-spatial: Geospatial Data Integration using GeoSPARQL-to-SQL Translation (*demo paper*). In Proceedings of the 15th International Semantic Web Conference (ISWC), Kobe, Japan, October 2016.

[ConverterToRdf] W3C. ConverterToRdf. https://www.w3.org/wiki/ConverterToRdf

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

[D2RQ] The D2RQ Mapping Language. http://d2rq.org/d2rq-language

[DKF+15] A. Dimou, D. Kontokostas, M. Freudenberg, R. Verborgh, J. Lehmann, E. Mannens, S. Hellmann, and R. Van de Walle. Assessing and Refining Mappings to RDF to Improve Dataset Quality. In Proceedings of the 14th International Semantic Web Conference (ISWC), pp. 133-149, Bethlehem, Pennsylvania, USA, 2015.

[DSC+14] A. Dimou, M. van der Sande, P. Colpaert, R. Verborgh, E. Mannens, and R. van de Walle. RML: A generic language for integrated RDF mappings of heterogeneous data. In Proceedings of the 7th Workshop on Linked Data on the Web (LDOW), Seoul, Korea, April 2014.

[DBpedia] DBpedia. http://dbpedia.org

[EPSG] European Petroleum Survey Group Geodetic Parameter Registry. http://www.epsg-registry.org/

[ESRIGeoDB] ESRI Inc. Personal GeoDatabases. http://desktop.arcgis.com/en/arcmap/latest/manage-data/administer-file-gdbs/personal-geodatabases.htm

[ESRIshp] ESRI Inc. Shapefile Technical Description. http://www.esri.com/library/whitepapers/pdfs/shapefile.pdf

[FMEworkbench] ESRI Inc. FME Workbench for ArcGIS Data Interoperability extension. http://www.esri.com/software/arcgis/extensions/datainteroperability/key-features/spatial-etl

[GDAL] GDAL/OGR - Geospatial Data Abstraction Library. http://www.gdal.org/

[GeoJSON] GeoJSON specification. http://geojson.org/

[GeoKettle] Spatialytics.ordg. GeoKettle Spatial ETL tool. http://www.spatialytics.org/projects/geokettle/

[GeoKnow] GeoKnow EU/FP7 project. http://geoknow.eu/

[GeoKnow-D2.2] GeoKnow EU/FP7 project. D2.2.1. Integration of Geospatial Databases. http://svn.aksw.org/projects/GeoKnow/Public/D2.2.1_Integration_of_Geospatial_Databases.pdf

[GeoNames] GeoNames geographical database. http://www.geonames.org/

[GeoRDF] GeoRDF Profile. http://www.w3.org/wiki/GeoRDF

[GeoTools] GeoTools library. Available from http://www.geotools.org/

[geo2rdf] geometry2rdf Utility. http://mayor2.dia.fi.upm.es/oeg-upm/index.php/en/technologies/151-geometry2rdf. Source code: https://github.com/boricles/geometry2rdf/tree/master/Geometry2RDF

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

[GPL3] GNU General Public License, version 3.0. http://www.gnu.org/licenses/gpl-3.0.html

[HABF14] F. Hamdi, N. Abadie, B. Bucher, and A. Feliachi. GeomRDF: A Geodata Converter with a Fine-Grained Structured Representation of Geometry in the Web. In Proceedings of the 2nd International Workshop on Geospatial Lined Data (GeoLD), 2014.

[HSTORE] PostgreSQL hstore. https://www.postgresql.org/docs/current/static/hstore.html

[IBM-DB2] IBM DB2 Spatial Extender. http://www.ibm.com/software/products/us/en/db2spaext/

[INSPIRE] INSPIRE Knowledge Base. https://inspire.ec.europa.eu/

[INSPIRE-Themes] INSPIRE Knowledge Base. Data Specifications - Themes. http://inspire.ec.europa.eu/Themes/Data-Specifications/2892

[Jena] Apache Jena project. https://jena.apache.org/

[JenaDoc] Jena RDF Model documentation. https://jena.apache.org/documentation/javadoc/jena/org/apache/jena/rdf/model/Model.html

[JenaStream] RDF Streams in Apache Jena. https://jena.apache.org/documentation/io/streaming-io.html

[JTS] JTS Topology Suite. https://github.com/locationtech/jts

[KVS+14] K. Kyzirakos, I. Vlachopoulos, D. Savva, S. Manegold and M. Koubarakis. GeoTriples: a Tool for Publishing Geospatial Data as RDF Graphs Using R2RML Mappings. In Proceedings of the 6th International Workshop on the Foundations, Technologies and Applications of the Geospatial Web (*Terra Cognita*), Riva del Garda, Trentino, Italy, October 2014.

[LGD] LinkedGeoData project: Adding a spatial dimension to the Web of Data. http://linkedgeodata.org

[MAVEN] Apache Maven. http://maven.apache.org/

[MSAccess] Microsoft Inc. Access DBMS. https://products.office.com/access

[MSSQLServer] Microsoft Inc. SQL Server. https://www.microsoft.com/sql-server

[mySQL] MySQL Database. Available from http://www.mysql.com/

[NeoGeo] NeoGeo Vocabulary Specification. http://geovocab.org/doc/neogeo

[OGC07] Open Geospatial Consortium Inc. OpenGIS Geography Markup Language (GML) Encoding Standard. Version 3.2.1, 27/08/2007. http://portal.opengeospatial.org/files/?artifact_id=20509

[OGC10] Open Geospatial Consortium Inc. OpenGIS Implementation Specification for Geographic information - Simple feature access- Part 2: SQL option. OpenGIS Implementation Standard. Version 1.2.1, 04/08/2010. http://portal.opengeospatial.org/files/?artifact_id=25354

[OGC12] Open Geospatial Consortium Inc. OGC GeoSPARQL standard - A geographic query language for RDF data. Version 1.0, 27/04/2012. https://portal.opengeospatial.org/files/?artifact_id=47664

[OGC-POI] OGC Points of Interest SWG. http://www.opengeospatial.org/projects/groups/poiswg

[Oracle] Oracle Inc. Oracle Spatial and Graph. http://www.oracle.com/technetwork/database-options/spatialandgraph/overview/

[OracleRDFViews] Oracle Inc. RDF Views: Relational Data as RDF. https://docs.oracle.com/en/database/oracle/oracle-database/12.2/rdfrm/rdf-views.html

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

[OSMOnto] OpenStreetMap Map Features. https://wiki.openstreetmap.org/wiki/Map_Features

[OsmPoisPbf] OsmPoisPbf tool. https://github.com/MorbZ/OsmPoisPbf

[Ozi] OziExplorer Waypoints format. http://www.oziexplorer3.com/eng/help/fileformats.html

[Parliament] Parliament High-Performance Triple Store. http://parliament.semwebcentral.org/

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

[PostgreSQL] PostgreSQL DBMS. http://www.postgresql.org/

[RR15] M. Rodriguez-Muro and M. Rezk. Efficient SPARQL-to-SQL with R2RML mappings. Journal of Web Semantics, 33: 141-169, 2015.

[SANSA] SANSA Semantic Analytics Stack. https://github.com/SANSA-Stack

[TripleGeo] Athena R.C. TripleGeo. Java source code: https://github.com/SLIPO-EU/TripleGeo

[R2RML] W3C. R2RML: RDB to RDF Mapping Language. https://www.w3.org/TR/r2rml/

[RDFDirect] W3C. A Direct Mapping of Relational Data to RDF. https://www.w3.org/TR/rdb-direct-mapping/

[SDI4Apps] SDI4Apps project. http://sdi4apps.eu/

[SLIPO-D2.1] SLIPO Project. Deliverable D2.1: POI Models and Identifiers Registry.

[Sponger] OpenLink Software. Virtuoso Sponger. http://vos.openlinksw.com/owiki/wiki/VOS/VirtSponger

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

[SpatiaLite] SpatiaLite. https://www.gaia-gis.it/fossil/libspatialite/index

[TTOverlay] TomTom Overlay files. http://www.tomtom.com/lib/doc/ttnavsdk3_manual.pdf

[UHAS12] J. Unbehauen, S. Hellmann, S. Auer, and C. Stadler. Knowledge Extraction from Structured Sources. Search Computing, pp. 34-52, Springer LNCS vol. 7538, 2012.

[Virtuoso] OpenLink Software. Virtuoso RDF Triple Store. https://virtuoso.openlinksw.com/rdf/

[VirtGeoRDF] Virtuoso RDF and Geometry. http://docs.openlinksw.com/virtuoso/rdfsparqlgeospat/

[VVS+10] L.M. Vilches-Blázquez, B. Villazón-Terrazas, V. Saquicela, A. de León, O. Corcho, and A. Gómez-Pérez. GeoLinked Data and INSPIRE through an Application Case. In Proceedings of the 18th SIGSPATIAL International Conference on Advances in Geographic Information Systems (ACM SIGSPATIAL GIS 2010), pp. 446-449, San Jose, California, USA, November 2010.

9. Annex

9.1. POI Transformation for the SLIPO pilots

TripleGeo ver.1.4 has been also applied against POI datasets provided by the SLIPO industrial partners, to be used in the pilot of WP5. Table 8 offers a preliminary overview of the original POI datasets and their resulting RDF representations; a detailed description of POI characteristics, their applicable mappings to the SLIPO ontology, as well as their diverse classification schemes will be included in Deliverable D5.1 "Specification of the Pilots".

Provider	Dataset	Spatial coverage	Input records	Output triples
WiGeoGIS	Herold	Austria	350,053	26,214,671
WiGeoGIS	Wigeoapi (OSM)	Austria	156,594	3,366,026
TomTom	MultiNet	Austria	311,078	13,587,632
TomTom	MultiNet	Germany	2,425,708	106,650,699
TomTom	MultiNet	Greece	127,910	4,998,524
GET	get-pois_v07	Greece	72,373	3,313,133
Athena RC	OpenStreetMap	Europe	7,447,693	121,439,699

Table 8: Datasets provided by SLIPO partners and transformed into RDF using TripleGeo ver.1.4

9.2. Configuration Settings

9.2.1. TripleGeo Configuration for Transformation

The following listing is an indicative configuration (`shp_options.conf`) for TripleGeo ver.1.4 in order to transform data from an ESRI shapefile into RDF triples. It can be applied with the following command, assuming that binaries are bundled together in `/target/triplegeo-1.4-SNAPSHOT.jar`:

```
java -cp ./target/triplegeo-1.4-SNAPSHOT.jar
eu.slipo.athenarc.triplegeo.Exractor ./test/conf/shp_options.conf
```

This configuration file contains the following properties (explanatory comments are given in green colour):

```
##Possible input formats: SHAPEFILE, DBMS, CSV, GPX, GEOJSON, XML, OSM

inputFormat = SHAPEFILE

##Transformation mode: specify either 'GRAPH' (on disk) or 'STREAM' (in-memory)
or 'RML' (for applying user-specified RML mappings)

mode = STREAM
```

```
##Paths to directories and files used by the application

## CURRENTLY SUPPORTED: You can specify MULTIPLE input files (of exactly the same format and attributes) separating them by ';' in order to activate multiple concurrent threads for their transformation.

tmpDir = ./tmp

inputFiles = ./test/data/points.shp

outputDir = ./test/output

## OPTIONAL parameter for the encoding (character set) for strings in the input data. If not specified, UTF-8 encoding is assumed.

encoding = UTF-8

##Possible export formats: RDF/XML, RDF/XML-ABBREV, N-TRIPLES, TURTLE (or TTL), N3

serialization = N-TRIPLES

## Specify the spatial ontology for geometries in the exported data.

## Possible values: 1) GeoSPARQL, 2) Virtuoso (legacy RDF ontology for points only), 3) wgs84_pos (for WGS84 Geoposition RDF vocabulary)

targetGeoOntology = GeoSPARQL

##File (in TTL or YAML format) specifying mappings of the input attribute schema to RDF properties; i.e., prescribing how input features will be transformed into RDF triples (typically according to an ontology).

mappingSpec = ./test/conf/poi_mappings.yml

## File (either in CSV or YAML format) containing a classification hierarchy in categories assigned to input features. Classification is only applied if a suitable mapping (including a category attribute) has been specified above. Leave blank if non applicable.

classificationSpec = ./test/poi_classification.csv

## Boolean specifying whether the data features specify their category based on its identifier in the classification scheme (false) or the actual name of the category (true).

classifyByName = true

## Attribute parameters (CASE-sensitive for shapefiles!!!)

attrKey = osm_id

attrGeometry = the_geom

attrName = name
```

```
attrCategory = type

registerFeatures = true

## MANDATORY parameter that specifies the data source provider of the input features

featureSource = OSM_sample_points

## OPTIONAL parameters regarding namespaces of generated URIs:

nsOntology = http://slipo.eu/def#

nsGeometry = http://www.opengis.net/ont/geosparql#

nsFeatureURI = http://slipo.eu/id/poi/

nsClassURI = http://slipo.eu/id/term/

nsClassificationURI = http://slipo.eu/id/classification/

nsDataSourceURI = http://slipo.eu/id/poisource/

## Specify two lists (of comma separated values) with the correspondence between a prefix

## and its respective namespace (mainly used in attribute mappings)

prefixes = slipo, geo, xsd, rdfs, wgs84_pos

namespaces = http://slipo.eu/def#, http://www.opengis.net/ont/geosparql#, http://www.w3.org/2001/XMLSchema#, http://www.w3.org/1999/02/22-rdf-syntax-ns#, http://www.w3.org/2003/01/geo/wgs84_pos#

## Spatial Reference parameters

## If not specified, geometries are assumed in WGS84 reference system (EPSG:4326).

sourceCRS = EPSG:2100

targetCRS = EPSG:4326

## OPTIONAL parameter. Default language tag for string literals created in the output RDF.

defaultLang = en
```

9.2.2. TripleGeo Configuration for Reverse Transformation

The following listing is an indicative configuration (`shp_reverse.conf`) for TripleGeo ver.1.4 in order to transform data from RDF triples back to records in an ESRI shapefile. It can be applied with the following command, assuming that binaries are bundled together in `/target/triplegeo-1.4-SNAPSHOT.jar`:

```
java -cp ./target/triplegeo-1.4-SNAPSHOT.jar
eu.slipo.athenarc.triplegeo.ReverseExtractor ./test/conf/shp_reverse.conf
```

This configuration file contains the following properties (explanatory comments are given in green colour):

```
##Possible output formats: SHAPEFILE, CSV

outputFormat = SHAPEFILE

##Paths to directories and files used by the application

## CURRENTLY SUPPORTED: You can specify MULTIPLE input RDF files (of the same
serialization) separating them by ';'.

inputFiles = ./test/output/poi_classification.nt;./test/output/points.nt

outputFile= ./test/output/points_reconstructed.shp

sparqlFile= ./test/conf/points_query.sparql

tmpDir = ./tmp

## OPTIONAL parameter for the encoding (character set) for strings in the output
data. If not specified, UTF-8 encoding is assumed.

encoding = UTF-8

##Possible serialization formats for input triples: RDF/XML, RDF/XML-ABBREV, N-
TRIPLES, TURTLE (or TTL), N3

serialization = N-TRIPLES

## Spatial Reference parameters

## If not specified, geometries are assumed in WGS84 reference system (EPSG:4326).

sourceCRS = EPSG:4326

targetCRS = EPSG:2100

## OPTIONAL property. Default language for the string literals used in the input
RDF

defaultLang = en
```

9.3. Sample Mappings

9.3.1. Sample RML Mapping

The following is a listing concerning an indicative RML Mapping (Section 3.3.2.1) for a POI dataset to the SLIPO ontology. For clarity, attribute names are shown in bold. Note that the URIs regarding classification to categories (`CATEGORY_URI`) are assigned on-the-fly according to a suitable classification scheme specified by the user.

```
@prefix rr: <http://www.w3.org/ns/r2rml#>.
```

```
@prefix rml: <http://semweb.mmlab.be/ns/rml#>.
@prefix ql: <http://semweb.mmlab.be/ns/ql#>.
@prefix slipo: <http://slipo.eu/def#>.
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.
@prefix rdfs: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix geo: <http://www.opengis.net/ont/geosparql#>.
@prefix sf: <http://www.opengis.net/ont/sf#>.
@prefix wgs84_pos: <http://www.w3.org/2003/01/geo/wgs84_pos#>.
<#POIMapping>
  rml:logicalSource [
    rml:source "" ;
    rml:referenceFormulation ql:CSV
  ];
  rr:subjectMap [
    rr:template "http://slipo.eu/id/poi/{UUID}";
    rr:class slipo:POI;
      rr:class geo:Feature
  ];
    rr:predicateObjectMap [
    rr:predicate slipo:name;
    rr:objectMap [
      rr:parentTriplesMap <#POIName>
        ]
    ];
   rr:predicateObjectMap [
    rr:predicate slipo:category;
    rr:objectMap [
      rr:template "http://slipo.eu/id/term/{CATEGORY_URI}";
        rr:class slipo:Term;
        ]
    ];
```

```
  rr:predicateObjectMap [
    rr:predicate slipo:lastUpdated;
    rr:objectMap [
      rml:reference "TIMESTAMP";
        rr:datatype xsd:dateTime
      ]
    ];
   rr:predicateObjectMap [
    rr:predicate geo:hasGeometry;
    rr:objectMap [
      rr:parentTriplesMap <#POIGeometry>
        ]
      ].
<#POIName>
  rml:logicalSource [
    rml:source "" ;
    rml:referenceFormulation ql:CSV
  ];
  rr:subjectMap [
    rr:template "http://slipo.eu/id/poi/{UUID}/name";
    rr:class slipo:Name
  ];
  rr:predicateObjectMap [
    rr:predicate slipo:nameLang;
    rr:objectMap [
      rr:constant "en"
      ]
    ];
  rr:predicateObjectMap [
    rr:predicate slipo:nameValue;
    rr:objectMap [
```

```
      rml:reference "NAME";
        rr:language "en";
        rr:termType rr:Literal
      ]
    ].
<#POIGeometry>
  rml:logicalSource [
    rml:source "" ;
    rml:referenceFormulation ql:CSV
  ];
  rr:subjectMap [
    rr:template "http://slipo.eu/id/poi/{UUID}/geometry";
    rr:class sf:Geometry
  ];
  rr:predicateObjectMap [
    rr:predicate geo:asWKT;
    rr:objectMap [
      rml:reference "WKT";
        rr:datatype geo:wktLiteral
      ]
    ]
```

9.3.2. Sample YAML Mapping

The following is a listing concerning an indicative YAML Mapping (Section 3.3.2.2) for a POI dataset to the SLIPO ontology. Note that attributes concerning UUIDs and geometries need not be specified in this mapping, as they are either constructed or recognized on-the-fly when TripleGeo works in GRAPH or STREAM transformation modes. For clarity, attribute names are shown in bold. Also note that the URIs regarding classification to categories (`CATEGORY_URI`) are assigned on-the-fly according to a suitable classification scheme specified by the user.

```
osm_id:
  partOf: sourceInfo
  entity: source
```

```
  predicate: slipo:poiRef
DATA_SOURCE:
  partOf: sourceInfo
  entity: source
  predicate: slipo:sourceRef
CATEGORY_URI:
  entity: category
  predicate: slipo:category
  datatype: uri
name:
  instanceOf: name
  entity: name
  predicate: slipo:name
  type: official
  language: en
timestamp:
  entity: lastUpdated
  predicate: slipo:lastUpdated
  datatype: timestamp
```

9.4. Sample Classification Schemes

9.4.1. Classification Hierarchy in CSV format

The following listing is an indicative classification scheme in CSV format with two levels (*category, subcategory*) applicable against POI data extracted from OpenStreetMap (OSM). In such a CSV file, each line (record) represents a subcategory, also specifying its respective (parent) category. For each named category or subcategory, their corresponding identifier is also given. In this CSV file, comma is used as the delimiter character between attributes, whereas string values are enclosed in double quotes (e.g., `"HOTEL"`). The header of the CSV file contains the names of the corresponding attributes.

```
"category_id","category","subcategory_id","subcategory"
1,"ACCOMMODATION",3,"CARAVAN"
1,"ACCOMMODATION",5,"HOSTEL"
1,"ACCOMMODATION",6,"HOTEL"
```

```
1,"ACCOMMODATION",7,"MOTEL"
1,"ACCOMMODATION",1,"ALPINEHUT"
1,"ACCOMMODATION",2,"CAMPING"
1,"ACCOMMODATION",4,"CHALET"
2,"AMENITY",8,"COURT"
2,"AMENITY",9,"FIRESTATION"
2,"AMENITY",11,"LIBRARY"
2,"AMENITY",12,"PLAYGROUND"
2,"AMENITY",13,"POLICE"
2,"AMENITY",14,"POSTOFFICE"
2,"AMENITY",15,"PRISON"
2,"AMENITY",16,"PUBLICBUILDING"
2,"AMENITY",17,"TOWNHALL"
3,"BARRIER",18,"BLOCKS"
4,"EDUCATION",19,"COLLEGE"
4,"EDUCATION",20,"NURSERY"
4,"EDUCATION",21,"SCHOOL"
4,"EDUCATION",22,"UNIVERSITY"
5,"FOOD",23,"BAR"
5,"FOOD",24,"BIERGARTEN"
5,"FOOD",25,"CAFE"
5,"FOOD",26,"FASTFOOD"
5,"FOOD",27,"ICECREAM"
5,"FOOD",28,"PUB"
5,"FOOD",29,"RESTAURANT"
6,"HEALTH",30,"DENTIST"
6,"HEALTH",31,"DOCTORS"
6,"HEALTH",32,"HOSPITALEMERGENCY"
6,"HEALTH",33,"HOSPITAL"
6,"HEALTH",34,"PHARMACY"
6,"HEALTH",35,"VETERINARY"
```

```
7,"LANDUSE",36,"ALLOTMENTS"
7,"LANDUSE",37,"CONIFEROUSDECIDUOUS"
7,"LANDUSE",38,"CONIFEROUS"
7,"LANDUSE",39,"DECIDUOUS"
7,"LANDUSE",40,"GRASS"
7,"LANDUSE",41,"HILLS"
7,"LANDUSE",42,"MILITARY"
7,"LANDUSE",43,"QUARY"
7,"LANDUSE",44,"SCRUB"
7,"LANDUSE",45,"SWAMP"
8,"MONEY",46,"BANK"
8,"MONEY",47,"EXCHANGE"
10,"POW",48,"BAHAI"
10,"POW",49,"BUDDHIST"
10,"POW",50,"CHRISTIAN"
10,"POW",51,"HINDU"
10,"POW",52,"ISLAMIC"
10,"POW",53,"JAIN"
10,"POW",54,"JEWISH"
10,"POW",55,"SHINTO"
10,"POW",56,"SIKH"
10,"POW",58,"UNKOWN"
9,"POI",59,"CAVE"
9,"POI",60,"CRANE"
9,"POI",61,"EMBASSY"
9,"POI",62,"BUNKER"
9,"POI",63,"MINE"
9,"POI",64,"PEAK1"
9,"POI",65,"PEAK"
9,"POI",66,"CITY"
9,"POI",67,"HAMLET"
```

```
9,"POI",68,"SUBURB"
9,"POI",69,"TOWN"
9,"POI",70,"VILLAGE"
9,"POI",71,"TOWERCOMMUNICATION"
9,"POI",72,"TOWERLOOKOUT"
11,"SHOP",73,"ALCOHOL"
11,"SHOP",74,"BAKERY"
11,"SHOP",75,"BICYCLE"
11,"SHOP",76,"BOOK"
11,"SHOP",77,"BUTCHER"
11,"SHOP",78,"CARREPAIR"
11,"SHOP",79,"CAR"
11,"SHOP",80,"CLOTHES"
11,"SHOP",81,"COMPUTER"
11,"SHOP",82,"CONFECTIONERY"
11,"SHOP",83,"CONVENIENCE"
11,"SHOP",84,"COPYSHOP"
11,"SHOP",85,"DEPARTMENTSTORE"
11,"SHOP",86,"DIY"
11,"SHOP",87,"FISH"
11,"SHOP",88,"FLORIST"
11,"SHOP",89,"GARDENCENTRE"
11,"SHOP",90,"GIFT"
11,"SHOP",91,"GREENGROCER"
11,"SHOP",92,"HAIRDRESSER"
11,"SHOP",93,"HEARINGAIDS"
11,"SHOP",94,"HIFI"
11,"SHOP",95,"JEWELRY"
11,"SHOP",96,"KIOSK"
11,"SHOP",97,"LAUNDRETTE"
11,"SHOP",98,"MARKETPLACE"
```

```
11,"SHOP",99,"PHONE"
11,"SHOP",100,"MOTORCYCLE"
11,"SHOP",101,"MUSIC"
11,"SHOP",102,"NEWSPAPER"
11,"SHOP",103,"PET"
11,"SHOP",104,"SHOES"
11,"SHOP",105,"SUPERMARKET"
11,"SHOP",106,"TOBACCO"
11,"SHOP",107,"TOYS"
11,"SHOP",108,"VENDINGMASCHINE"
11,"SHOP",109,"VIDEORENTAL"
12,"SPORT",110,"ARCHERY"
12,"SPORT",111,"BASEBALL"
12,"SPORT",112,"BASKETBALL"
12,"SPORT",113,"BOWLING"
12,"SPORT",114,"CANOE"
12,"SPORT",115,"CRICKET"
12,"SPORT",116,"DIVING"
12,"SPORT",117,"FOOTBALL"
12,"SPORT",118,"GOLF"
12,"SPORT",119,"GYM"
12,"SPORT",120,"GYMNASIUM"
12,"SPORT",121,"CLIMBING"
12,"SPORT",122,"HORSE"
12,"SPORT",123,"ICESKATING"
12,"SPORT",124,"LEISURECENTER"
12,"SPORT",125,"MINIATURGOLF"
12,"SPORT",126,"MOTORRACING"
15,"WATER",169,"DAM"
12,"SPORT",127,"SHOOTING"
12,"SPORT",128,"SKATING"
```

```
12,"SPORT",129,"SKIINGDOWNHILL"
12,"SPORT",130,"SNOOKER"
12,"SPORT",131,"SOCCER"
12,"SPORT",132,"STADIUM"
12,"SPORT",133,"SWIMMING"
12,"SPORT",134,"TENNIS"
12,"SPORT",135,"WATERSKI"
12,"SPORT",136,"SURFING"
13,"TOURIST",137,"ARCHAELOGICAL"
13,"TOURIST",138,"ART"
13,"TOURIST",139,"ATTRACTION"
13,"TOURIST",140,"BATTLEFIELD"
13,"TOURIST",141,"BEACH"
13,"TOURIST",142,"CASTLE"
13,"TOURIST",143,"CASTLE2"
13,"TOURIST",144,"CINEMA"
13,"TOURIST",145,"FOUNTAIN"
13,"TOURIST",146,"INFORMATION"
13,"TOURIST",147,"MEMORIAL"
13,"TOURIST",148,"MONUMENT"
13,"TOURIST",149,"MUSEUM"
13,"TOURIST",150,"NIGHTCLUB"
13,"TOURIST",151,"RUINS"
13,"TOURIST",152,"THEATRE"
13,"TOURIST",153,"THEMEPARK"
13,"TOURIST",156,"WINDMILL"
13,"TOURIST",157,"WRECK"
13,"TOURIST",158,"ZOO"
14,"TRANSPORT",159,"TERMINAL"
14,"TRANSPORT",160,"AIRPORT"
14,"TRANSPORT",161,"BUSSTOP"
```

```
14,"TRANSPORT",162,"FUEL"
14,"TRANSPORT",163,"LIGHTHOUSE"
14,"TRANSPORT",164,"MARINA"
14,"TRANSPORT",165,"RENTALCAR"
14,"TRANSPORT",166,"SUBWAY"
14,"TRANSPORT",167,"STATION"
14,"TRANSPORT",168,"TRAMSTOP"
15,"WATER",170,"TOWER"
15,"WATER",171,"WEIR"
```

9.4.2. Classification Hierarchy in YAML Format

The aforementioned classification scheme for OSM POI data can be alternatively specified in YAML format. Such a user-prepared YAML file has indentations to denote breakdown of a given *category* (shown in bold) into *subcategories* (i.e., two blank characters in the beginning of a line at each extra level in the hierarchy). The identifier of each category (at any level) is specified after its name and it is preceded with a '#' character.

```
ACCOMMODATION #1
  HOSTEL #5
  HOTEL #6
  MOTEL #7
  ALPINEHUT #1
  CAMPING #2
  CHALET #4
AMENITY #2
  COURT #8
  FIRESTATION #9
  LIBRARY #11
  PLAYGROUND #12
  POLICE #13
  POSTOFFICE #14
  PRISON #15
  PUBLICBUILDING #16
  TOWNHALL #17
```

```
BARRIER #3
  BLOCKS #18
EDUCATION #4
  COLLEGE #19
  NURSERY #20
  SCHOOL #21
  UNIVERSITY #22
FOOD #5
  BAR #23
  BIERGARTEN #24
  CAFE #25
  FASTFOOD #26
  ICECREAM #27
  PUB #28
  RESTAURANT #29
HEALTH #6
  DENTIST #30
  DOCTORS #31
  HOSPITALEMERGENCY #32
  HOSPITAL #33
  PHARMACY #34
  VETERINARY #35
LANDUSE #7
  ALLOTMENTS #36
  CONIFEROUSDECIDUOUS #37
  CONIFEROUS #38
  DECIDUOUS #39
  GRASS #40
  HILLS #41
  MILITARY #42
  QUARY #43
```

SCRUB #44

SWAMP #45

MONEY #8

BANK #46

EXCHANGE #47

POW #10

BAHAI #48

BUDDHIST #49

CHRISTIAN #50

HINDU #51

ISLAMIC #52

JAIN #53

JEWISH #54

SHINTO #55

SIKH #56

UNKOWN #58

POI #9

CAVE #59

CRANE #60

EMBASSY #61

BUNKER #62

MINE #63

PEAK1 #64

PEAK #65

CITY #66

HAMLET #67

SUBURB #68

TOWN #69

VILLAGE #70

TOWERCOMMUNICATION #71

TOWERLOOKOUT #72

SHOP #11

ALCOHOL #73

BAKERY #74

BICYCLE #75

BOOK #76

BUTCHER #77

CARREPAIR #78

CAR #79

CLOTHES #80

COMPUTER #81

CONFECTIONERY #82

CONVENIENCE #83

COPYSHOP #84

DEPARTMENTSTORE #85

DIY #86

FISH #87

FLORIST #88

GARDENCENTRE #89

GIFT #90

GREENGROCER #91

HAIRDRESSER #92

HEARINGAIDS #93

HIFI #94

JEWELRY #95

KIOSK #96

LAUNDRETTE #97

MARKETPLACE #98

PHONE #99

MOTORCYCLE #100

MUSIC #101

NEWSPAPER #102

```
  PET #103
  SHOES #104
  SUPERMARKET #105
  TOBACCO #106
  TOYS #107
  VENDINGMASCHINE #108
  VIDEORENTAL #109
SPORT #12
  ARCHERY #110
  BASEBALL #111
  BASKETBALL #112
  BOWLING #113
  CANOE #114
  CRICKET #115
  DIVING #116
  FOOTBALL #117
  GOLF #118
  GYM #119
  GYMNASIUM #120
  CLIMBING #121
  HORSE #122
  ICESKATING #123
  LEISURECENTER #124
  MINIATURGOLF #125
  MOTORRACING #126
WATER #15
  DAM #169
  SHOOTING #127
  SKATING #128
  SKIINGDOWNHILL #129
  SNOOKER #130
```

```
  SOCCER #131
  STADIUM #132
  SWIMMING #133
  TENNIS #134
  WATERSKI #135
  SURFING #136
TOURIST #13
  ARCHAELOGICAL #137
  ART #138
  ATTRACTION #139
  BATTLEFIELD #140
  BEACH #141
  CASTLE #142
  CASTLE2 #143
  CINEMA #144
  FOUNTAIN #145
  INFORMATION #146
  MEMORIAL #147
  MONUMENT #148
  MUSEUM #149
  NIGHTCLUB #150
  RUINS #151
  THEATRE #152
  THEMEPARK #153
  WINDMILL #156
  WRECK #157
  ZOO #158
TRANSPORT #14
  TERMINAL #159
  AIRPORT #160
  BUSSTOP #161
```

```
FUEL #162
LIGHTHOUSE #163
MARINA #164
RENTALCAR #165
SUBWAY #166
STATION #167
TRAMSTOP #168
TOWER #170
WEIR #171
```

9.5. Sample SPARQL Query for Reverse Transformation

The following SPARQL query can be applied on an RDF graph regarding POI data extracted according to the SLIPO ontology. This query may be employed for reconstructing records with specific attributes (shown in bold) as declared in the SELECT clause:

```
PREFIX slipo: <http://slipo.eu/def#>
PREFIX geo: <http://www.opengis.net/ont/geosparql#>
PREFIX sf: <http://www.opengis.net/ont/sf#>
SELECT ?osm_id ?shape ?type ?name ?timestamp
WHERE {
            OPTIONAL {
                  ?uri slipo:source ?poiSourceInfo .
                  ?poiSourceInfo slipo:poiRef ?osm_id .
            }
            OPTIONAL {
                  ?uri slipo:category ?category .
                  ?category slipo:value ?type .
            }
            OPTIONAL {
                  ?uri geo:hasGeometry ?geometry .
```

```
            ?geometry geo:asWKT ?shape .
        }
        OPTIONAL {
            ?uri slipo:name ?fName .
            ?fName slipo:nameType "official" .
            ?fName slipo:nameValue ?name .
        }
        OPTIONAL {
            ?uri slipo:lastUpdated ?timestamp .
        }
    }
```