Collaborative Project

GeoKnow – Making the Web an Exploratory for Geospatial Knowledge

Project Number: 318159       Start Date of Project: 01/12/2012       Duration: 36 months

Deliverable 3.2.1
Fusing of geographic features

<table>
<thead>
<tr>
<th>Dissemination Level</th>
<th>Public</th>
</tr>
</thead>
<tbody>
<tr>
<td>Due Date of Deliverable</td>
<td>Month 12, 30/11/2013</td>
</tr>
<tr>
<td>Actual Submission Date</td>
<td>29/11/2013</td>
</tr>
<tr>
<td>Work Package</td>
<td>WP 3, Spatial knowledge aggregation, fusing &amp; quality assessment</td>
</tr>
<tr>
<td>Task</td>
<td>T 3.2</td>
</tr>
<tr>
<td>Type</td>
<td>Report</td>
</tr>
<tr>
<td>Approval Status</td>
<td>Final</td>
</tr>
<tr>
<td>Version</td>
<td>1.0</td>
</tr>
<tr>
<td>Number of Pages</td>
<td>48</td>
</tr>
<tr>
<td>File name</td>
<td>D3.2.1_Fusing_of_geographic_features.pdf</td>
</tr>
</tbody>
</table>

Abstract: This document describes FAGI, a framework designed to support a wide functionality for fusing geospatial RDF data. First, a brief review of data fusion in general and its application on the individual areas of geospatial data and RDF data is provided. Next, specificities and challenges on fusing geospatial RDF data are identified and the architecture of FAGI framework is described. Finally, the first two implemented tools of the framework are presented: FAGI-tr for transforming RDF representations of geospatial data, and FAGI-gis that provides the infrastructure for processing and transforming geometries for fusing geometric features of linked entities.

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

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Reason</th>
<th>Revised by</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>13/11/2013</td>
<td>First draft of the deliverable</td>
<td>Giorgos Giannopoulos, Thomas Maroulis</td>
</tr>
<tr>
<td>0.1</td>
<td>14/11/2013</td>
<td>Improvements on the draft</td>
<td>Dimitris Skoutas</td>
</tr>
<tr>
<td>0.2</td>
<td>15/11/2013</td>
<td>Improvements on the draft</td>
<td>Spiros Athanasiou</td>
</tr>
<tr>
<td>0.3</td>
<td>17/11/2013</td>
<td>Version prepared for the project’s peer-review</td>
<td>Giorgos Giannopoulos</td>
</tr>
<tr>
<td>0.4</td>
<td>22/11/2013</td>
<td>Peer review</td>
<td>Mohamed Sherif</td>
</tr>
<tr>
<td>0.5</td>
<td>25/11/2013</td>
<td>Improvements on peer review comments</td>
<td>Giorgos Giannopoulos, Dimitris Skoutas, Spiros Athanasiou</td>
</tr>
<tr>
<td>0.1</td>
<td>29/11/2013</td>
<td>Final version</td>
<td>Giorgos Giannopoulos</td>
</tr>
</tbody>
</table>
## Author List

<table>
<thead>
<tr>
<th>Organisation</th>
<th>Name</th>
<th>Contact Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATHENA</td>
<td>Spiros Athanasiou</td>
<td><a href="mailto:spathan@imis.athena-innovation.gr">spathan@imis.athena-innovation.gr</a></td>
</tr>
<tr>
<td>ATHENA</td>
<td>Giorgos Giannopoulos</td>
<td><a href="mailto:giann@imis.athena-innovation.gr">giann@imis.athena-innovation.gr</a></td>
</tr>
<tr>
<td>ATHENA</td>
<td>Thomas Maroulis</td>
<td><a href="mailto:thomasmaroulis@gmail.com">thomasmaroulis@gmail.com</a></td>
</tr>
<tr>
<td>ATHENA</td>
<td>Dimitris Skoutas</td>
<td><a href="mailto:dskoutas@imis.athena-innovation.gr">dskoutas@imis.athena-innovation.gr</a></td>
</tr>
</tbody>
</table>
### Time Schedule before Delivery

<table>
<thead>
<tr>
<th>Next Action</th>
<th>Deadline</th>
<th>Care of</th>
</tr>
</thead>
<tbody>
<tr>
<td>First version</td>
<td>13/11/2013</td>
<td>Giorgos Giannopoulos (ATHENA)</td>
</tr>
<tr>
<td>First consolidated version</td>
<td>17/11/2013</td>
<td>Giorgos Giannopoulos (ATHENA)</td>
</tr>
<tr>
<td>Peer review</td>
<td>22/11/2013</td>
<td>Mohamed Sherif (InfAI)</td>
</tr>
<tr>
<td>Submit final restructured version</td>
<td>29/11/2013</td>
<td>Giorgos Giannopoulos (ATHENA)</td>
</tr>
</tbody>
</table>
Executive Summary

This document describes a framework, called FAGI, for fusing different RDF representations of geometries into a consistent map. First, we provide background knowledge on data fusion, focusing particularly on state of the art methodologies for RDF data fusion, as well as geospatial data fusion. Next, we present the fusion directions we propose in this framework, taking into account existing work on fusion and a preliminary analysis on geospatial RDF datasets. The architecture of the envisioned framework for fusing geospatial RDF data is described; then, we present the currently implemented fusion tools, namely:

- **FAGI-tr**, a tool for recognizing different RDF representations of spatial features in RDF (i.e., different vocabularies, different literal (feature values) formats and different coordinate reference systems) and transforming these representations from one to another.
- **FAGI-gis**, a tool for performing geospatial transformations on RDF geometry features, so that they can be used on complex fusion facilities, where two or more geometries need to be changed and combined into a fused one.

The layout of the document is the following.

In Chapter 1, we provide background knowledge on data fusion. First, we define the fusion process and present some generic fusion objectives and strategies. Second, we present existing work on RDF data fusion, as well as on geospatial data fusion.

In Chapter 2, based on the conclusions derived from the survey of previous works, we point out some general practices followed in the literature, as well as some important shortcomings of current approaches, w.r.t. geospatial RDF fusion. Then, presenting the specificities of the fusion problem on geospatial RDF data, we identify open issues and challenges, and specify directions for investigation. Subsequently, we present a high level architecture of the fusion framework FAGI, along with a brief description of the functionality of its components.

In Chapter 3, we present the currently implemented fusion tools (components) of the FAGI framework that handle the transformation of different RDF representations of geometries (in terms of vocabulary, geometry serializations and coordinate reference system) and we provide the infrastructure for performing geospatial transformations (e.g. scaling, rotation of geometries), when necessary in the fusion process.

In Chapter 4, we conclude with some general remarks on the implemented tools and we discuss the next steps, i.e. the next components to be implemented in the FAGI framework.
## Abbreviations and Acronyms

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRS</td>
<td>Coordinate Reference System</td>
</tr>
<tr>
<td>DBMS</td>
<td>DataBase Management System</td>
</tr>
<tr>
<td>GIS</td>
<td>Geographical Information Systems</td>
</tr>
<tr>
<td>GML</td>
<td>Geography Markup Language</td>
</tr>
<tr>
<td>OGC</td>
<td>Open Geospatial Consortium (<a href="http://www.opengeospatial.org/">http://www.opengeospatial.org/</a>)</td>
</tr>
<tr>
<td>RDF</td>
<td>Resource Description Framework</td>
</tr>
<tr>
<td>SPARQL</td>
<td>SPARQL Protocol and RDF Query Language</td>
</tr>
<tr>
<td>SQL</td>
<td>Structured Query Language</td>
</tr>
<tr>
<td>WGS84</td>
<td>World Geodetic System (EPSG:4326)</td>
</tr>
<tr>
<td>WKT</td>
<td>Well Known Text (as defined by ISO 19125)</td>
</tr>
</tbody>
</table>
# Table of Contents

1. **INTRODUCTION** ......................................................................................................................... 10
   1.1 **DATA FUSION** .......................................................................................................................... 10
       1.1.1 Definition .............................................................................................................................. 10
       1.1.2 Fusion strategies ..................................................................................................................... 11
   1.2 **RDF DATA FUSION** .................................................................................................................. 13
       1.2.1 Sieve ....................................................................................................................................... 13
       1.2.2 ODCleanStore ...................................................................................................................... 13
       1.2.3 KnoFuss ............................................................................................................................... 14
       1.2.4 ALOE ............................................................................................................................... 14
   1.3 **GEOSPATIAL DATA FUSION** .................................................................................................... 14

2. **FUSION FRAMEWORK** .................................................................................................................. 17
   2.1 **PRINCIPLES AND SHORTCOMINGS** ..................................................................................... 17
       2.1.1 Fusion of geospatial data ....................................................................................................... 17
       2.1.2 Fusion of RDF data ............................................................................................................... 17
   2.2 **SPECIFICITIES AND DIRECTIONS** ....................................................................................... 18
   2.3 **FRAMEWORK ARCHITECTURE** .............................................................................................. 19

3. **DEVELOPED TOOLS** .................................................................................................................... 23
   3.1 **FAGI-TRANSFORMATION** ........................................................................................................ 23
       3.1.1 Tool description ..................................................................................................................... 24
       3.1.1.1 Implementation information .............................................................................................. 24
       3.1.1.2 Components .................................................................................................................... 24
       3.1.1.3 Supported data sources and formats .................................................................................... 25
       3.1.2 Usage manual ....................................................................................................................... 26
       3.1.2.1 Installation/Building instructions ....................................................................................... 26
       3.1.2.2 Configuration file ............................................................................................................. 26
       3.1.2.3 Demonstration scenario .................................................................................................... 28
       3.1.3 Licensing ............................................................................................................................. 30
       3.1.4 Future work ............................................................................................................................ 31
   3.2 **FAGI-GEOSPATIAL PROCESSING** ........................................................................................... 32
       3.2.1 Tool description ..................................................................................................................... 32
       3.2.1.1 Implementation information .............................................................................................. 32
       3.2.1.2 Components .................................................................................................................... 33
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.2.1.3</td>
<td>Supported data sources and formats</td>
<td>35</td>
</tr>
<tr>
<td>3.2.2</td>
<td>Manual</td>
<td>35</td>
</tr>
<tr>
<td>3.2.2.1</td>
<td>Installation/Building instructions</td>
<td>35</td>
</tr>
<tr>
<td>3.2.2.2</td>
<td>Usage - Demonstration scenario</td>
<td>35</td>
</tr>
<tr>
<td>3.2.3</td>
<td>Licensing</td>
<td>38</td>
</tr>
<tr>
<td>3.2.4</td>
<td>Future work</td>
<td>38</td>
</tr>
</tbody>
</table>

4. CONCLUSION | 39
5. REFERENCES | 40
6. APPENDIX | 42

6.1 BASIC FAGI-TR CLASSES | 42
6.2 BASIC FAGI-GIS CLASSES | 44
6.3 POSTGRES DB SCHEMA USED BY FAGI-GIS | 46
List of Figures

Figure 1: Fusion Framework Architecture ................................................................................... 21
Figure 2: FAGI-tr components ........................................................................................................ 25
Figure 3: Dataset loading panel ...................................................................................................... 29
Figure 4: Rule matching panel ......................................................................................................... 30
Figure 5: Dataset transformation panel ........................................................................................ 31
Figure 6: FAGI-gis components .................................................................................................... 34
Figure 7: Postgres DB configuration ............................................................................................. 35
Figure 8: Dataset loading and geospatial features indexing ............................................................ 36
Figure 9: Geometry similarity scoring and fusion action selection ................................................. 37

List of Tables

Table 1: Conflict resolution strategies ............................................................................................. 11
1. Introduction

In this chapter, we provide background knowledge on data fusion. First, we describe the fusion problem, as well as the data integration problem in general. Next, we present existing works on the individual areas of RDF data fusion and geospatial data fusion (conflation).

1.1 Data fusion

1.1.1 Definition

Fusion is the process of merging the descriptions (attributes, metadata, properties) of two or more resources that correspond to the same real world entity, to produce a richer, cleaner and universal description for the respective entity. Fusion constitutes the final part of the data integration process, which usually consists of three steps: schema integration, duplicate detection and fusion [BN08].

The first step aims to align the schemata used to describe different datasets in order to facilitate the identification and merging of entities in the next steps of the process. This can be achieved either by schema integration or by schema mapping. In the former, a universal, optimal schema is defined based on the individual schemata with the aim to integrate them. In the latter case, the aim is to create mappings from source schemata to a single target schema. The objective of this step is to reduce the search space of the problem, by limiting the comparison of resources between different datasets, only to those resources that are characterized by same/similar schema elements. For example, given two datasets A and B, that contain information about public buildings and are described by respective ontologies \( O_a \) and \( O_b \), in order to identify the same schools in the two datasets, one would have to compare every resource of dataset A to every resource of dataset B, which is a costly operation. Instead, if the respective ontologies include classes for schools and this information can be captured by schema mapping/integration, then the comparisons can be restricted only to pairs of entities belonging to the respective classes.

The second step, duplicate detection, aims at identifying and linking resources (from different datasets, or even within the same dataset) that correspond to the same real world entities. It is also known as record linkage, deduplication, object identification, reference reconciliation, link discovery, interlinking etc. The output of the process is usually a unified identifier that describes the individual duplicates or a link between the entities (e.g. owl:sameAs link). The specific problem is computationally challenging, since, in the naive scenario, all entities of dataset A must be compared to all entities of dataset B, resulting to an \( O(n^2) \) complexity which is prohibitive when the compared datasets contain hundreds of thousands or even millions of records. Several methodologies and algorithms have been proposed to increase the efficiency of the process, such as blocking or clustering algorithms [NH10, IJB11] or more sophisticated techniques applying sampling and considering geometric bounds for the identification of duplicates [Ngo12]. In order for the individual entities to be compared, several similarity functions can be applied (cosine, jaccard, levenshtein, etc.) [NH10] both on the identifiers/names of the entities and on the rest of their attributes, given that (a) they are mapped after the first step, and (b) it is meaningful to apply similarity functions on their values in order to identify duplicate entities. In the aforementioned example, the aim of this step is to produce links between schools of datasets A and B that are identified to correspond to the same real world school.

The third step handles the merging of the linked entities, that is the production, for each set of linked entities, of a richer, more correct and more complete description, w.r.t. to the attributes describing the
properties. It involves recognizing which attributes/properties of the entities correspond to each other and resolving potential conflicts or irregularities, such as different values or domains for the same properties, lack of values or properties, differences in the quality of datasets, etc. The main challenge of this task is to be able to effectively apply the most appropriate conflict resolution/merging strategy, w.r.t. to the properties and values to be used. Back to our example, given two interlinked schools, one of their properties might describe the geographic location of the buildings. Supposing that dataset A is known to be the most precise dataset w.r.t. building locations, the best strategy would be to keep the geographic location values of dataset A and discard the respective values of dataset B. On the other hand, if another property enumerates the names of the teachers of the school, a suitable strategy might be to keep the union of the values coming from dataset A and B, since some teacher names might be missing from either of the datasets.

Next, we focus on data fusion, presenting a categorization of fusion strategies as discussed in [BN08].

1.1.2 Fusion strategies

The fusion process can be divided into two phases: (a) identifying the properties of the interlinked entities that correspond to each other and the conflicts in their respective values, and (b) deciding how to handle conflicting values. The first phase can be handled by applying similar mapping and matching methods to the previously described ones. Here, the task is less challenging than the problems of schema mapping and interlinking in the sense that the search space is much smaller (i.e. the properties of two interlinked entities). However, the second phase, which is deciding how to handle a conflict is not trivial and depends on several factors, such as quality measures on the datasets, recency, differences in the domains and the ranges of the compared property values, problem setting/context, as well as the preferences of the users.

Bleisher et al. [BN08] divide such strategies into three categories: conflict ignoring, conflict avoiding and conflict resolution strategies. Strategies of the first category do not resolve conflicts or they do not even recognize them. An example of such strategies would be presenting all conflicting values to the user to decide which one to keep. Conflict avoiding strategies resolve conflicts, but in a uniform manner, without considering individual cases. For example, one such strategy always prefers property values from a specific dataset. Finally, conflict resolution strategies handle conflict cases separately, by considering all available information of the entities to be fused. Table 1 presents several types of conflict resolution strategies as categorized in [BN05].

<table>
<thead>
<tr>
<th>Strategy</th>
<th>Functionality</th>
</tr>
</thead>
<tbody>
<tr>
<td>Count</td>
<td>Counts the number of distinct non-null values, i.e., the number of conflicting values. Only indicates conflicts, the actual data values are lost.</td>
</tr>
<tr>
<td>Min / Max</td>
<td>Returns the minimal/maximal input value with its obvious meaning for numerical data. Lexicographical (or other) order is needed for non numerical data.</td>
</tr>
<tr>
<td>Sum / Avg / Median</td>
<td>Computes sum, average and median of all present non-null data values. Only applicable to numerical data.</td>
</tr>
<tr>
<td>Variance / Stddev</td>
<td>Returns variance and standard deviation of data values. Only applicable to numerical data.</td>
</tr>
<tr>
<td>Random</td>
<td>Randomly chooses one data value among all non-null data values.</td>
</tr>
<tr>
<td>Choose</td>
<td>Returns the value supplied by a specific source.</td>
</tr>
<tr>
<td>Coalesce</td>
<td>Takes the first non-null value appearing.</td>
</tr>
<tr>
<td>First / Last</td>
<td>Takes the first/last value of all values, even if it is a null value</td>
</tr>
<tr>
<td>------------------------------</td>
<td>---------------------------------------------------------------------</td>
</tr>
<tr>
<td>Vote</td>
<td>Returns the value that appears most often among the present values. Ties can be broken by a variety of strategies, e.g., choosing randomly.</td>
</tr>
<tr>
<td>Group</td>
<td>Returns a set of all conflicting values. Leaves resolution to the user.</td>
</tr>
<tr>
<td>Shortest / Longest</td>
<td>Chooses the value of minimum/maximum length according to a length measure.</td>
</tr>
<tr>
<td>Concat</td>
<td>Returns the concatenated values. May include annotations, such as source of value.</td>
</tr>
<tr>
<td>Highest Quality</td>
<td>Evaluates to the value of highest information quality, requiring an underlying quality model.</td>
</tr>
<tr>
<td>Most Recent</td>
<td>Takes the most recent value. Most recentness is evaluated with the help of another attribute or other data about recentness of tuples/values.</td>
</tr>
<tr>
<td>Most Active</td>
<td>Returns the most often accessed or used value. Usage statistics of the DBMS can be used in evaluating this function.</td>
</tr>
<tr>
<td>Choose Corresponding</td>
<td>Chooses the value that belongs to the value chosen for another column.</td>
</tr>
<tr>
<td>Most complete</td>
<td>Returns the value from the source that contains the fewest null values in the attribute in question.</td>
</tr>
<tr>
<td>Most distinguishing</td>
<td>Returns the value that is the most distinguishing among all present values in that column.</td>
</tr>
<tr>
<td>Highest information value</td>
<td>According to an information measure this function returns the value with the highest information value.</td>
</tr>
<tr>
<td>Most general / specific concept</td>
<td>Using a taxonomy or ontology this function returns the most general or specific value.</td>
</tr>
</tbody>
</table>

Each of the above strategies can be applied separately on groups of interlinked entities, by defining restrictions that assign a strategy to a specific group of linked entities, according to the features of the respective interlinked entities. An example of such restrictions is described in Section 1.1.1, where one of the restrictions confines the application of a strategy only to the properties containing the names of the teachers of the school entities. In this specific case, the selected strategy could be Concat, which returns a concatenated literal containing values from the respective properties of both entities that are interlinked.

Moreover, a combination of strategies may be applied by creating rules that examine several features of the interlinked entities and apply accordingly one or more consecutive resolution functions [BN06]. Finally, the above strategies can be incorporated into a learning model that correlates features of the interlinked entities (and their properties) with conflict strategies and automatically selects the most fitting strategy. An example of such process would be a supervised classification algorithm that trains a classification model based on previous, manual selections of strategies for interlinked entities and utilizes this model to propose the most suitable strategy. For example, such a model could be trained on a consistent user behavior, where properties with categorical values are constantly concatenated, and apply concatenation on every future property with such values. [BN08] gives a thorough categorization of data integration systems that handle the whole process of data integration, including data fusion. The systems are categorized according to several factors: the data models they adopt/support, the mapping and duplicate detection techniques, the conflict resolution strategies they support, etc. From this categorization it is obvious that there is a lack of approaches that automate the fusion process based on trained classification models: most automated approaches involve conflict avoiding strategies, where a fusion strategy is applied uniformly to all sets of interlinked entities.
1.2 RDF data fusion

Although there have been several works on schema integration/mapping and interlinking on RDF data, fusion is still a field of ongoing research. Current approaches adopt state of the art, generic fusion techniques, without specializing them based on the characteristics of the RDF data model and schemata. Besides, none of the existing works propose methodologies for automating the fusion process on large RDF datasets. Next, we briefly describe the latest approaches handling fusion on RDF data.

1.2.1 Sieve

The work described in [MPM12] introduces the Sieve framework for quality assessment and fusion of Linked Data, that is part of a larger framework for Linked Data integration (LDIF [SMI+12]). The fusion functionality offered by Sieve is inspired by state of the art work on data fusion ([IBN05]).

The quality assessment process, among others, facilitates the fusion process by providing quality scores on the data, so that these scores can be used by a fusion strategy. It takes into account factors such as timeliness of data, provenance, as well as user configurable preference lists on features of the data or the dataset as a whole.

The fusion process is defined through XML configuration files, where the user can specify: (a) the classes of the objects to be considered for fusion, (b) the properties to be fused and (c) for each property, the fusion function to be applied. Sieve supports the following functions (strategies):

- **Filter**: removes all values for which the input quality assessment metric is below a given threshold.
- **KeepFirst**: keeps the value with the highest value for a given quality assessment metric. In case of ties, the function keeps the first in order of input.
- **KeepAllValuesByQualityScore**: similar to KeepFirst, but in case of ties, it keeps all values with the highest score.
- **Average**: takes the average of all input data for a given property.
- **Voting**: picks the value that appeared most frequently across sources. Each named graph has one vote, the most voted value is chosen.
- **WeightedVoting**: picks the value that appeared most frequently across highly rated sources. Each named graph has one vote proportional to its score for a given quality metric, the value with highest aggregated scores is chosen.

1.2.2 ODCleanStore

Another framework that includes fusion functionality on RDF data is ODCleanStore [MK12, KMN12]. The framework supports linking, cleaning, transformation and quality assessment operations on Linked Data. The fusion component of the framework supports several user configurable conflict resolution strategies for fusion, that also consider provenance and quality metadata of the datasets:

- **ANY, MIN, MAX, SHORTEST, LONGEST**: An arbitrary value, minimum, maximum, shortest, or longest is selected from the conflicting values V
- **AVG, MEDIAN, CONCAT**: Computes the average, median, or concatenation of conflicting values
- **BEST**: The value with the highest aggregate quality is selected
- **LATEST**: The value with the newest time is selected
- **ALL**: All input values are preserved
1.2.3 Knofuss

Through Knofuss framework [NVM+08], a system for interlinking, conflict detection and fusion is implemented. The presented work focuses on interlinking (coreferencing). It implements several variations of the Jaro-Winkler string similarity metric and an adaptive learning clustering algorithm, which are applied in a configurable way, depending on the ontology classes the respective entities belong to.

1.2.4 ALOE

The tool described in [ALOE, Ngo12b], is the first tool, to the best of our knowledge, to handle automatic fusion of RDF data. ALOE supports automatic discovery of class and property mappings across endpoints even when no schema information is available. Moreover, it applies machine learning techniques to learn transformation/fusion rules for string values in several levels (i.e., characters, n-grams, words). It offers two learning modes: batch and active learning. In the first case, it utilizes a training set on which the algorithm learns rules, while in the second case, the algorithm learns the rules during an interactive, step by step process, where the user provides feedback to the model. ALOE is currently still under development, however, experiments performed on its effectiveness [Ngo12b] yield encouraging results w.r.t. the task of automatically fusing RDF properties with textual content.

1.3 Geospatial data fusion

In the domain of geospatial data management, fusion (or conflation) is the process of merging two different geospatial datasets that cover overlapping regions, in such way that the best quality elements of the individual datasets are kept in the final, composite dataset [CK08]. The conflation process may be applied to any combination of vector (i.e., points, lines, polygons) and raster data (i.e., images). Next, we focus only to fusion approaches applicable on vector data. In this setting, the attributes to be fused contain the geometries of the respective spatial entities (e.g., the coordinates of the points that form a polygon).

Geospatial data fusion has been an area of study for several decades, thus, there is a considerable amount of approaches handling the problem. In general, the process of fusing geospatial objects comprises the following steps:

1. **Feature matching.** Find a set of conjugate point pairs, termed control point pairs, in two datasets. That is, identify the points within the two datasets, which most probably correspond to each other, so that these points can be used as reference points for the rest of the fusion process.
2. **Match checking.** Enhance the set of control point pairs by filtering out pairs with ambiguous quality. This process can also utilize non-spatial metadata of the objects in order to produce an as accurate as possible set of control point pairs.
3. **Spatial attribute alignment.** Use the accurate control points to align the rest of the geospatial objects (e.g., points or lines) in both datasets by using space partitioning and transformation techniques (e.g., scaling or “stretching” geometries until they are aligned).

One of the earliest works on the field proposes an automatic, iterative process for conflating geospatial objects [Saa88]. The process iteratively identifies new matching points (control point pairs) and transforms geometries, until no new point pairs can be identified. The steps of the process are described below:

1. Define test criteria that will be used to identify matching point pairs from the two datasets. These criteria might consider geometric properties of the objects, such as close coordinates, in-out-degree of points, direction, or non-spatial metadata of the objects such as name, address, etc.
2. Combine test criteria to perform matching. This process is handled specially at the first iteration, since the purpose then is to identify the most accurate point pairs. So, the focus is on strict matching based on geospatial proximity. On the contrary, during the next iterations, the datasets have undergone transformations that have aligned the objects to some degree, so the focus then moves to less strict criteria that examine the overall similarity of objects.

3. Apply criteria and produce matching point pairs.

4. Align matching points by applying space transformation. This process first partitions the space through triangulation based on the matching points and then transforms each partition through a rubber-sheeting transformation, which essentially stretches the space of the two datasets until they are aligned.

5. Repeat the process starting from step 2, if more matching point pairs can be identified.

Several geospatial matching criteria, as well as the triangulation method [PS85] are further analysed in the above work. The evaluation of the method demonstrates fairly good accuracy of matching, pointing out the trade-off between false-positive and false-negative results.

The work of [WF99] also proposes a multi-step, but not iterative process for matching spatial objects. The approach focuses on fusing streets (lines) and utilizes several statistics of the spatial objects as matching criteria. Briefly, the steps of the process are the following:

1. Preprocessing, which incorporates the processes of control points identification and space transformation, so that global errors of the datasets (e.g. topology, digitization errors, missing elements) are minimized.
2. Identification of potential matching geometry pairs. This step utilizes the concept of buffers that are created around candidate matching elements and guide the matching.
3. Geometric constraint application. This step filters out of the list of matching object pairs, all the pairs that do not satisfy strict geometric restrictions, considering angle, length and distance of the elements.
4. Evaluation of matching pairs. The list of matching pairs is further refined with a merit function that utilizes statistics of the dataset.
5. Calculation of the final matchings. In this step, the search space is partitioned into smaller parts, so that the matchings of the objects can be efficiently evaluated.

The evaluation of the method produces satisfying matching results, requiring, however, prohibitively large execution times. Finally, the specific work discusses the need for definition and application of quality measures, as part of a postprocessing step of the described process.

The work of [WJ98] adopts a similar approach, in the sense that it considers buffers around spatial objects and examines containment relations between objects (edges between points). The process consists in feature matching and feature updating. During feature matching, if necessary, edges are divided in such way that full containment between edges and/or edge parts is achieved. During updating, the matched edges are replaced with weighted averages and are combined/connected with edges of the datasets that could not be matched at the first step.

The authors of [CCM+98] base their proposed approach on the Vector Product Format (VPF), a standard for organizing geospatial data, including thematic information (i.e. categories of spatial objects), as well as topological relations between them (e.g. neighbouring objects). Feature matching is performed by combining spatial and thematic criteria, i.e. both geospatial proximity and similarity (in feature descriptions and categories) are taken into account. For feature alignments, triangulation and rubber-sheeting
techniques are applied. A final deconfliction step is applied, where a unified feature is produced out of the individual matching features, using quality metadata stored in the VPF model.

A more recent approach described in [CSK+06] focuses on conflating road networks. Road intersections are utilized as control points, based on previous works indicating that intersection points are good candidates for feature matching. Then, for each of the datasets to be fused, the distribution of the intersection points is analysed, and feature matching is performed on intersection point sets. Several optimizations are applied to reduce the number of point sets to be compared, utilizing distances, angles and degrees (considering points as graph nodes) of intersection points. The method achieves high precision and recall values.

In [SA11], the authors discuss the problem of geospatial data fusion as a means to increase data quality. They focus their description on improving the quality of road networks by fusing roundabouts. To this end, they exploit the degrees of the points of the roundabout and the angles between the road segments. The same authors, at [SA09], present algorithms as well as a tool for fusing geospatial features. The main focus is first on data preprocessing (decomposition of polyline based datasets, extraction of branches) and merging of geometrical and semantic information (branch assignment, integration of geodata). Their evaluation shows high accuracy; however, the authors point out some shortcomings of the proposed methods (e.g. applicability only on restricted search space).
2. Fusion Framework

In this chapter, we first discuss some basic principles and shortcomings of the current works for fusing RDF data and geospatial data. Based on these, we identify a series of challenges to be fulfilled by an integrated fusion system for geospatial RDF data. Finally, we present the architecture of the envisioned fusion system, describing briefly the functionality of each component and its importance for the fusion process.

2.1 Principles and shortcomings

2.1.1 Fusion of geospatial data

The problem of fusing (conflating) geospatial features (in our case geometric shapes represented as points, lines and polygons) has been studied for several decades. A common ground in several proposed methods is the identification of control point pairs that, according to a scoring function, are the most probable to represent the same objects between the two datasets to be fused. The identification of these control point pairs is usually based on strict geospatial matching (utilizing distances, angles, lengths, etc.) since the accuracy of the resulting pairs influences the rest of the fusion process.

Another widely adopted practice is the partition of space into sub-areas, usually applying triangulation based on the control points. This process serves as a pre-processing step, so that a space transformation (e.g. affine transformations, rubber-sheeting) is applied to one of the datasets, or both of them, in order for the rest of the geometries to be aligned to each other, following the alignment of the control points.

Some of the proposed methods utilize a combination of spatial and non-spatial criteria in matching/scoring functions, others use statistics on geospatial properties and relations in the datasets, while certain methods are iterative and others not. Finally, several methods work (semi)automatically, either transforming geometries, or producing new geometries to replace the fused ones.

On the other hand, the major drawback of most methods seems to be their very low efficiency. This is mainly due to the very complex geospatial operations that are required in several steps of the process: normalization of the datasets, calculation of spatial relations such as proximity and angles, space transformations, etc. Another issue is the reduction of the search space, which is the number of comparisons to be made between objects of the two compared datasets. To this end, partition, clustering or candidate items filtering techniques are applied, without, however, significantly improving the efficiency of the methods. Another common drawback is the restricted utilization of semantic, non-spatial properties of the geospatial features, which can contribute both by increasing the accuracy of the feature matching process, and by reducing the search space. This is a predictable shortcoming, though, since most methods in the literature where dedicated on processing conventional geospatial data, lacking the semantic richness of geospatial RDF datasets.

2.1.2 Fusion of RDF data

Work on fusing RDF datasets is quite limited, since most works on RDF data integration focus on schema mapping and interlinking of entities. The few works handling fusion adopt existing fusion concepts and strategies, without extending them to exploit the inherent characteristics of RDF data. Also, none of the proposed approaches tries to automate the fusion process, that is, to apply supervised or unsupervised
training techniques to recommend fusion strategies on specific datasets or parts of them, without the need for manual configuration of the strategy from the user.

2.2 Specificities and Directions

The problem of fusing geospatial RDF datasets differentiates, to some extent, from the individual settings of fusing geospatial data, or fusing RDF data. This results from the fact that both semantic and spatial knowledge is available and needs to be taken into account when performing the fusion process. A main difference lies in the following. In the case of RDF data, there is a clear and explicit separation between the two steps of interlinking and fusion: first, matching entities from the two input datasets are identified and linked together, and then their attributes are fused to produce a single entity representation. In the case of geospatial data, matching and fusing are rather intertwined (e.g., identifying control points that are used to drive the fusion of geometries). Thus, when dealing with RDF entity pairs from two compared datasets, along with their geospatial representations, the additional knowledge that these pairs correspond to the same real world object, and thus, their geometries correspond to each other, is a given. Still, the fusion process in this case needs to handle more complex sets of attributes, including strings or arithmetic values in conjunction with geometries of entities, which in certain scenarios requires transformations/combinations of existing geometries. So, a potential set of fusion actions/strategies involves the following:

- Define effective scoring functions that meaningfully combine semantic properties and geospatial features of the entities. These scoring functions should utilize:
  - Available mappings on the schemas that characterize the interlinked entities.
  - Refined string similarity functions that can decompose the names and descriptions of the entities and apply individually fitting similarity metrics. For example, given the names "Joe's diner" and "8th elementary school of Athens" of a restaurant and a school respectively, numeric values ("8th") as well as special words ("diner", "elementary") should be treated separately when calculating the similarity between interlinked entities.
- Select the most accurate geometry according to some criterion. For example, in some cases the most accurate geometry might be considered the most complex one (i.e. the one consisting of the most points), or the most recent one. In other cases, the most accurate geometry is given explicitly by the quality indicators of the datasets, if available.
- Select a combination of geometries that produces a richer geometry. This strategy may involve several actions/sub-strategies. We present some indicative examples:
  - Keep partial geometric properties from both geometries, to produce a new geometry of higher quality. For example, in an interlinked pair of geometries, if geometry A is a point with high quality coordinates and the other geometry is polygon B with less qualitative coordinates, then the fused geometry should be the polygon B, shifted so that its new centroid becomes the point A. In another case, where two points are interlinked, the final geometry could be produced by averaging the coordinates of the two points.
  - Select and keep both geometries separately. In this case, the geometries could, indeed, correspond to the same real world entity but represent slightly different aspects of this entity. For example, in the case of two interlinked entities, geometry A may represent the building of a school, while geometry B might represent both the school building and the school yard. In this case, a decision on how to handle these two entities should be made:
    - Keep both geometries for the entity, exactly as they are in separate properties.
- Create a new, composite geometry object and use it as the geometry of the fused entity.
- Create a new entity and keep two entities corresponding to the semantics of the two separate geometries as result of the fusion.
- Reject the link between two entities (清除). In case the matching score of two interlinked geometries is too low, there might be the case that these two geometries have been falsely interlinked.
- Produce new links between entities. Even if there are no links between two entities, after examining clusters/neighborhoods of interlinked entities, and by applying traditional geospatial conflation techniques (space transformation, statistics on spatial properties, etc), new links between entities of the two datasets may be recognized. This makes the problem even more challenging, since it increases both the complexity of specifying the rules for the fusion process and its computational complexity.
- A further action that can be part of the fusion process is, apart from fusing the attributes of the matched entities, to produce an overall fused dataset that contains entities from both initial datasets. This entails a process similar to the one described above: by searching on neighborhoods of interlinked entities, and concluding that an entity of dataset B does not exist in dataset A, the entity can be created in A too, and be interlinked with the respective entity of dataset B.
- Produce new semantic, topological relations. Based on analysis of the geospatial relations between interlinked entities some basic geospatial relations between entities could be calculated and stored as semantic metadata of the entities. For example, properties that capture, for each entity, the orientation (e.g. north from) of its top-k closest neighbors could be created.

Another specificity of our problem setting, compared to previous works, is the fact that, due to the already existing links between entities, automatic conflation techniques that process the whole space of the datasets cannot be straightforwardly applied. On the other hand, these interlinked entities can be utilized as control point pairs for partitioning the space and working on each partition separately.

Finally, a great challenge is to implement meaningful learning algorithms that will be able to train on user actions and recommend both fusion scores and preferred fusion strategies. A critical factor for the effectiveness of such algorithms is the proper selection of training features, i.e. features that describe interlinked entities/geometries pairs and correlate them to the user selected fusion strategy. The algorithms should function both offline and online. That is, the algorithms should be able to be trained offline on previous user actions, or online, as the user manually selects a fusion action for a (set of) interlinked entity pairs.

### 2.3 Framework Architecture

In this section, we describe the envisioned framework for fusing geospatial RDF data called FAGI - Fusion and Aggregation for Geospatial Information. FAGI aims at providing a comprehensive solution for geospatial enrichment of knowledge bases, based on semantic technologies and open data sources. Its main functionality is to support the entire process of aligning RDF representations of geospatial data, fusing and aggregating geospatial triples through intuitive, step-by-step processes, model-learning for automatic fusion recommendations and map-based UIs. Our focus is to build the FAGI framework based on real world and diverse geospatial RDF datasets, so that we can capture the heterogeneity of data sources and provide an extended processing coverage to several use case scenarios.

Figure 1 presents the architecture of the framework. The basic components are as follows.
• **Storage Component.** This component is essentially an RDF store where data and metadata from all stages of the fusion process are stored and retrieved. This is where the input data are initially stored and the new, processed data, metadata from the Learning Component, as well as the final fused dataset(s) are written.

• **Preprocessing Component.** This is the component that performs an initial processing of the data, so that they are normalized w.r.t. the vocabularies they use (i.e. RDF vocabularies for representation of geospatial features, OGC standards for the encoding of geometries in the form of literals, coordinate reference systems). Also, this component gathers and organizes quality metadata of the datasets, which either exist within the datasets, or are manually input by the user.

• **Geospatial Processing Component.** This component is assigned with the whole process of geospatial indexing, topological relations calculation and geospatial transformations of the geometric features of the entities. Geospatial indexing is necessary for being able to perform efficiently the remaining two processing categories. The functionality of the other two modules is self-evident: Geospatial calculations are required in order for geometric similarity scores for fusion are produced, while geospatial transformation operations are necessary for the actual fusing process, when a geometry or even a whole set of geometries need to be transformed to produce a final, qualitative fused geometry.

• **Fusion Component.** This component performs the core fusion functionality of the framework. It includes property mapping for both spatial and non-spatial properties, calculation of similarity scores for geospatial features of entities and recommendation of fusion strategies. Furthermore, based on the calculated similarity scores (both on spatial and non-spatial properties of the interlinked entities), as well as on geospatial calculations on the neighborhoods of the interlinked entities, it suggests creation of new links between entities or rejection of existing links, as false positives.

• **User Interface Component.** This component supports the interaction of the end user with the rest, underlying components. Through this component, the user can configure and guide the preprocessing of the input datasets, select and apply fusion strategies on (parts of) the datasets and provide feedback either by validating the fusion results or by validating the new link creation or the link rejection recommendations.

• **Learning Component.** This component handles the training of machine learning models for automatic suggestion of fusion strategies for groups of triples, based on the individual characteristics of the interlinked geospatial entities, as well as the history of user actions w.r.t. selecting and validating fusion actions.
Each component of Figure 1 is coloured based on the framework's development phase they correspond too. At this first iteration (green coloured components), we focused on creating the infrastructure for fusing different representations of geometry in RDF. This involves both aligning the vocabularies, the geometry serializations and the CRS of geometries (Vocabulary Alignment Module) and implementing the infrastructure for indexing, comparing and transforming geometries to be fused (Geospatial Processing Component). Of course, this development iteration included also the development of parts of the general use Modules, such as the Selection of Fusion Actions Interfaces, the Storage Component and the Fusion Actions component, that correspond to the core components of the first iteration. We note that the general use (light-blue coloured) modules regard all phases of the framework’s development and will be constantly adapted and extended, following the development of the rest modules. The second development phase consists in implementing Property Mapping functionality and exploiting non-spatial RDF properties (metadata) in combination with spatial features, in order to enhance the fusion process. The third phase of
the frameworks development regards fusion w.r.t relational information of spatial entities. In this iteration, we will include the functionality of Link Discovery, Link Rejection and Relations Fusion Modules, that will handle the respective fusion challenges described in Section 2.2.
3. Developed tools

In this chapter, we present the first two implemented tools of the fusion framework and describe their functionality. These tools implement a large part of the functionality of Preprocessing and Geospatial Processing Components. The first tool, FAGI-tr (FAGI for transformations), handles the recognition of different RDF representations of spatial features in RDF, that is different vocabularies, literal (feature values) formats and coordinate reference systems, and the transformation of these representations from one to another. The second tool FAGI-gis (FAGI for geospatial processing), implements the infrastructure for performing geospatial transformations on RDF geometry features, so that they can be used on complex fusion facilities, where geometries much change or be combined in order to be fused. The next sections describe the functionality and the architecture of the tools, as well as their usage by showcasing example use case scenarios. Also, we describe the future steps regarding the development of the tools.

3.1 FAGI-transformation

Although GeoSPARQL [OGC12] is being established as the most promising standard for representing geospatial data as RDF data and querying them with the SPARQL query language, there is still high diversity in the formats and vocabularies used to represent geospatial features in RDF, such as GeoOWL, GeoRDF, GeoSPARQL etc. [GeoKnowD21], or even ad hoc vocabularies. Although these vocabulary differences may be ignored in the interlinking stage by schema mapping, they must be resolved at the fusion stage: in order to create a cleaner and richer representation of the fused data, the final outcome must follow a single vocabulary.

FAGI-tr provides a framework for (a) loading a source and a target geospatial RDF dataset, (b) identifying vocabularies for representing geospatial RDF data, (c) selecting, from both datasets, the representations to be considered for processing, (d) selecting a target vocabulary and transforming all geospatial triples from both datasets into the respective format and (e) outputting the two datasets (aligned with respect to their geospatial vocabularies) for further processing.

The tool provides a graphical user interface for executing the above functionality. It takes as input SPARQL endpoints from where source and target datasets are loaded and stored into the underlying RDF store, Virtuoso [Virtuoso]. Next, the two datasets are parsed, and preconfigured regular expressions that recognize different RDF representations of triples involving geospatial data are applied. The regular expressions are organized in distinct configuration files that are described in detail in Section 3.1.2.2. Currently, the matching rules are manually written by the user directly into the configuration files. However, a graphical user interface will be implemented in the next iteration of the tool, so that users can define new matching rules in a “trial and error” manner. For each dataset, the identified vocabularies are presented to the user in order to select the types of triples (that is the respective vocabularies) that she wants to further process; the rest of the triples are discarded. At the final stage, the user selects a target vocabulary (from all the available/defined vocabulary matching rules) and all non-discarded geospatial triples from both datasets are transformed into the respective vocabulary. The output is written either on the same datasets or new datasets can be created, so that the original ones are kept for future use.

In what follows, we describe the tool components, the rule matching configuration and we demonstrate the usage of the tool.
3.1.1 Tool description

3.1.1.1 Implementation information

**FAGI-tr** is implemented in Java, as a desktop application, and it comprises a graphical user interface for executing all the steps of the datasets transformation process. It uses Virtuoso RDF store as a back-end for storing the initial datasets and for outputting the final, transformed datasets. Furthermore, it utilizes five configuration files that formalize the vocabulary matching rules that are used both to identify and transform RDF representations of geospatial data (see Section 3.1.2.2). **FAGI-tr** also uses the following external libraries:

- **Apache Jena**, a Java framework for building Semantic Web applications [Jena]. It provides a collection of tools and Java libraries for developing semantic web and linked-data apps, tools and servers.
- **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).
- **Google Guava**. The Guava project contains several of Google’s core libraries that we rely on in our Java-based projects: collections, caching, primitives support, concurrency libraries, common annotations, string processing, I/O, and so forth.
- **Java Topology Suite (JTS)**. The JTS Topology Suite [JTS] is an API of 2D spatial predicates and functions, conforming to the OGC Simple Features Specification for SQL. JTS is open source (under the LGPL license) and provides a complete implementation of fundamental 2D spatial algorithms written in Java.

3.1.1.2 Components

**FAGI-tr** consists of four basic components, as depicted in Figure 2: The **GUI component** that provides the interface to the user to access the rest of the components, the **CORE component** that performs the rule matching and handles non-spatial RDF triples, the **GEOMETRY component** that provides general geometry handling functionality (i.e. parsing, transforming) and the **RULES component** that handles the management of vocabulary matching rules (i.e. parsing, validation, translation into SPARQL queries). Next, we describe in more detail the functionality of each component.

- **GUI component**. It consists of three interfaces, implementing a three step transformation process. The first interface allows the user to load the datasets to be processed, providing information about the respective SPARQL endpoints and graph URIs. It also takes as input the Virtuoso RDF store connection information and the matching rules configuration. The second interface presents to the user the rule matching results and allows her to select the set of triples to further process, by selecting the respective vocabulary rules, for both datasets. The third interface allows the user to select a target rule, defining RDF vocabulary, geometry serialization and coordinate reference system (if applicable) to transform the retained geospatial triples.
- **CORE component**. This component is responsible for fetching both conventional (non-spatial) triples and identified spatial triples from a source dataset and storing them into Virtuoso. It also handles vocabulary rule matching and provides matching metadata (e.g. number of matched geospatial triples to a specific vocabulary rule).
- **GEOMETRY component**. This component implements all the necessary geospatial functionality: It provides parsing and transformation functions for handling (a) geometry serializations (i.e. representation of the actual geometry values) and (b) coordinate reference systems (CRS) (i.e.
the reference model according to which the coordinates of the geometries have been calculated).

- **RULES component.** This component handles the synthesis of vocabulary matching rules in form of regular expressions and their translation into SPARQL queries to be applied on the RDF datasets both for matching vocabularies and for transforming from one vocabulary to another.

![Figure 2: FAGI-tr components](image)

In Section 6.1 of the Appendix, a more detailed description of the basic functionality of each component's class is provided.

### 3.1.1.3 Supported data sources and formats

Currently, **FAGI-tr** supports loading data from SPARQL endpoints where the actual endpoint and the graph URI of the dataset are required. The supported RDF triples format is N-triples. The output of the tool is written into the underlying Virtuoso RDF store. As far as RDF vocabularies for geospatial features representation are concerned, we currently have defined rules for two vocabularies: Basic Geo Vocabulary [GeoPos84] and GeoSPARQL. This gives in total six defined rules for the several variations of these vocabularies. Implementation of support of RDF files as input/output sources, several RDF triples formats (RDF/XML, Turtle, etc.) and definition of more vocabulary matching rules are part of ongoing work. We note that, in contrast with the first two goals, defining new rules is not a development task; it can straightforwardly be realized by defining proper regular expressions into the configuration files of the tool.
3.1.2 Usage manual

3.1.2.1 Installation/Building instructions

This initial version of FAGI-tr is publicly available, including the entire Java source code as well as a JAR package that contains executable binaries [FAGI-tr]. The JAR package can be run directly, given that Java JRE (or SDK) 1.7 (or later) are installed and a Virtuoso store is set up on the machine that the tool runs. FAGI-tr has been successfully tested on both MS Windows and Linux environments. The tool's code is organized in a Maven Project that can be directly loaded into an IDE (e.g. NetBeans) and be further developed.

3.1.2.2 Configuration file

Rules for matching and transforming triples are expressed in the form of triple restrictions and are defined in five separate configuration files. Before we revise each configuration rule type separately, we present some concepts that are essential for the definition of matching rules.

- `<UID>`: UIDs are used for the identification of a specific rule. They must be bracketed by '<', '>', and may contain the characters defined by the following regex '([a-z][0-9])'; they must be unique within each configuration file and finally must be prefixed by 'X_' where 'X' is a single character unique for each configuration file that indicates the type of rules defined within.

- `<URI>`: URIs are bracketed by '{', '}'. They are used directly in the SPARQL queries that are produced from the respective rule instead of being matched via regex.

- `<regex>`: Regular expressions must be bracketed by double quotes "". When producing the SPARQL queries, each regex is assigned a variable of the form '?var#' where # is a number and an entry in the following format will be added to the restriction: 'FILTER (REGEX (?var#, regex, ""))'. The 'r' flag indicates that matching is case insensitive.

- `<datatype>`: Datatypes are either of type `<URI>` and follow the same rules or have the special value 'none' which means no datatype matching will be attempted. They are only used in object rules.

- `<type>`: Type is a special value that tells the parser how to parse the geometry serialization contained in a literal. They are only used in object rules and can have the following values:
  - `&lat` : latitude
  - `&long` : longitude
  - `&lat_long` : lat, long pair (comma or space separated)
  - `&alt` : altitude (currently ignored by the geometry parsers and writers)
  - `&wkt` : Well Known Text
  - `&gml` : Geometry Markup Language

- `<variable>` & `<anon>`: Variables must begin with '?' and cannot begin with '?var'. Anonymous nodes must begin with '_'. They are only used in full triple rules.

Next, we present the four types of rules that we define. The first three rule types (property, class, object) are helper rules intended to improve the readability and formulation of full triple rules. The only rules that will be matched are the full triple rules. The values of any helper rule will be substituted into the full triple rule internally by the parser wherever the respective helper rule's UID is referenced.

**property**: The configuration file 'property' is used for the definition of rules that match RDF properties. Rules are defined in the following format:
where `<UID>` for property rules must be prefixed with 'p_' and `<regex OR URI>` can be either a URI or a regex that will be used to match a particular property. An example of this rule type is the following expression:

```
<p_wgs84_lat> {http://www.w3.org/2003/01/geo/wgs84_pos#lat}
```

that maps `http://www.w3.org/2003/01/geo/wgs84_pos#lat` property to `p_wgs84_lat` identifier.

**class**: The configuration file ‘class’ is used for the definition of rules that match classes. Similarly, rules are defined by the following syntax:

```
<UID> <regex OR URI>
```

where `<UID>` for class rules must be prefixed with 'c_' and `<regex OR URI>` can be either a URI or a regex that will be used to match a particular class. An example of this rule type is the following expression:

```
<c_geosparql_geometry> {http://www.opengis.net/ont/geosparql#Geometry}
```

**object**: The configuration file 'object' is used for the definition of rules that match object literals. Rules are defined by the following syntax:

```
<UID> <regex> <datatype> <type>
```

where `<UID>` for object rules must be prefixed with 'o_'. `<regex>` is a regex that will be used to match the object literal, `<datatype>` is used to match the literal datatype or will be ignored if set to 'none' and `<type>` is used to indicate to the geometry parser how to process the literal. An example of this rule type is the following expression:

```
<o_wkt>
".*\((point|linestring|polygon|polyhedralsurface|triangle|tin|multipoint|multilinestring|multipolygon|geometrycollection)(\szm|sz|\sm|)\s*\{.*\}*)"

none
&wkt
```

that matches a literal describing one of the geometries in the regular expression, serialized in the Well Know Text standard.

**full triple rules**: The configuration files 'triple_default' and 'triple_user' are used for the definition of the full triple rules that will be used for matching. These rules reference property, object and class rules. 'triple_default' contains predefined rules, while 'triple_user' contains user added rules. Both files use the same syntax and are handled the same way internally. Rules are defined by the following syntax:

```
<uid> <description> <number of triples> <triples> .
```

where the following apply:

- `<UID>` for full triple rules must be prefixed with 'd_' if they are default rules or 'u_' if they are user rules.
• <description> is used to add a human readable explanation of the rule's purpose. It must be enclosed in double quotes "". The string may contain no white spaces. Instead any '_' characters will be replaced with white spaces during parsing.
• <number of triples> is the number of triples this rule contains. If there is only one triple the entire rule can be in one line. Otherwise this value must be the last value in the first line and each triple must be in its own separate line.
• <triples> are used to define a restriction for matching specific graphs. The last triple in the rule must end in a '.'. Each triple is defined by the following syntax:
  o <variable OR anon> <regex OR UID of "property" rule OR URI> <regex OR UID of "object" rule OR "class" rule OR anon node>

Furthermore, for a rule to be valid, the following must apply: (a) There must be only one variable. This variable is assumed to define the root node to which the matched literals belong, (b) All other internal nodes must instead be defined as anonymous nodes and (c) There is no limit to the number of triples defined or to the number of different object rules referenced by those triples. However, each rule must only define a single type of geometry. If the same rule contains objects with more than one logical geometry type there is no guarantee which one the parser will use. (&lat and &long form a single logical geometry type and the parser will expect to find both or neither).

Below, two examples of full triple rules are presented:

```xml
<k_geosparql_wkt> ""
1
?x
"("http://www.opengis.net/ont/geosparql#asWKT"|"http://www.opengis.net/ont/geosparql#hasSerialization")" <o_wkt> .

<k_w3c_loc2> ""
3
?x <p_wgs84_loc> _:a
  _:a <p_wgs84_lat> <o_lat>
  _:a <p_wgs84_long> <o_long> .
```

### 3.1.2.3 Demonstration scenario

Next, we demonstrate the usage of the tool through the graphical user interface. The first screen, depicted in Figure 3, requires the user's input regarding the datasets to be processed and the Virtuoso store to be used to store them. For both source and target datasets, the SPARQL endpoint and the graph URI of the dataset are required. Optionally, the user can restrict the triples to be loaded from the dataset, by inserting a regular expression that restricts the subjects of the respective triples. Also, in case the datasets are remote, that is they are not contained in the user configured Virtuoso store, then the user can define the graph URIs within her own Virtuoso store, so that the fetched datasets can be stored into two distinctive graphs each: one graph storing the initial dataset and a second graph that stores the eventually transformed dataset. In case these options are not selected, the transformations are stored in the initial, local Virtuoso dataset graph, for each dataset.

Another option for the user is to load the links file containing `owl:sameAs` links between the entities of the two datasets, so that only triples regarding the specific interlinked entities will be fetched and transformed. If left empty, every resource from the datasets will be fetched. Finally, the user is required to input the Virtuoso store connection information.
After the user has set the required input, she can run the rule matching process for source and target dataset, using the interface presented in Figure 4. In the left panel of the screen, all the available rules are shown. Upon selecting one of them, the user can see whether the rule was matched and, if matched, with how many triples. Also, metadata for the matched rule are presented, such as its description, the structure of the rule, and a sample matching set of triples from the dataset.

The user is able to select which kind of triples to retain for further processing, that is, retain only the triples matching a specific vocabulary rule. This functionality allows the user to keep only certain vocabulary versions of the geospatial triples, saving processing effort of the next steps, as well as clearing out possible erroneously contained triples.
Finally, the user is presented with the fetching and transforming pane, where rule matching statistics for source and target dataset are presented (Figure 5). The user is then able to choose a target vocabulary rule, so that all retained geospatial triples from both datasets are transformed according to the vocabulary of the rule. The transformed triples, along with the unmodified non-spatial triples are then written into Virtuoso, in different output graphs, depending on the user selection on the dataset loading panel. There is also limited support for changing the CRS, which will be extended in future steps.

3.1.3 Licensing

The FAGI-tr tool is a free software and its current version (including the Java source code and sample data) is available from [FAGI-tr]. 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.
3.1.4 Future work

Although the presented, first version of the tool is fully functional and already tested in several geospatial RDF datasets, there are several aspects of the tool that are being extended, in order to increase its functionality and simplify the RDF representation transformation process for the end users.

As already mentioned, our priority is extending the input/output functionality, so that datasets can be loaded from and written to RDF files and, also, several RDF triple formats are supported. Another enhancement we intend to add is a better triple filtering mechanism, so that the user can fetch only RDF triples that follow specific criteria, dictated by a SPARQL query.

As far as the rule matching process is concerned, adding vocabulary rules in the form of regular expressions in configuration files is straightforward. However, we intend to also provide a graphical interface, so that users can manually synthesize regular expression rules and try them on test datasets on real time in order to result to complete vocabulary rules through a "trial and error" process.

On extending the transformation functionality, we intend to enrich the support of CRS transformation. Finally, we intend to replace the desktop GUI with a web based one, as soon as sufficient back-end functionality has been implemented.
3.2 FAGI-geospatial processing

The challenging part of the fusion process is the efficient and effective combination of geometries, so that richer and more accurate fused geometries are produced. The main goal is to be able to combine fusion similarity scores coming from non-spatial and spatial RDF properties of the interlinked entities and recommend fitting fusion strategies for the respective entities. FAGI-gis provides the infrastructure for (a) indexing geospatial features (b) computing similarities between geometries and (c) applying geometry fusion strategies, either by combining or by transforming geometries, implementing an initial set of functions that can be straightforwardly extended.

Since complex geospatial operations might become prohibitively time consuming, we based this initial implementation of the tool on a PostgreSQL/PostGIS database, which provides efficient geospatial indexing and a wide range of efficient calculation and transformation functions. Our goal is, when a large part of the presented framework is implemented and tested on real datasets, so that the extent of geospatial processing functionality is finalized, to replace the relational DBMS with either (a) Virtuoso RDF store, if its extended geospatial processing functionality (currently ongoing work) suffices against the framework’s demands or (b) with a more lightweight geospatial processing library, that suffices, however, in terms of efficiency, such as Java Topology Suite [JTS, JCS].

The implemented tool provides a graphical user interface for executing the above functionality in a step by step process. At first, the user is required to provide connection information for the PostGIS database and, then, input SPARQL endpoints and graph URIs for the source and target datasets, exactly as performed in FAGI-tr. Then, the geospatial features are extracted from the triples and stored/indexed in PostgreSQL/PostGIS tables. At the final step of the process, the user loads the links between the interlinked entities of the two datasets and calls the scoring function, to assess similarities between linked entities. Then, according to the calculated scores, the user can select which entity pairs to fuse and which fusion strategy to follow. Currently, we have implemented seven fusion actions but, most importantly, we have implemented the underlying infrastructure, so that several more geospatial fusion strategies can be implemented through the proper interfaces. The geospatial fusion actions utilize the functionality of PostgreSQL/PostGIS, by calling the respective functions and working on indexed geospatial features. The final outcome is a set of fused geometries, for the respective linked entity pairs, that are written back to Virtuoso RDF store.

3.2.1 Tool description

3.2.1.1 Implementation information

FAGI-gis is implemented in Java, as a desktop application, and it provides a graphical user interface for executing all the steps of the datasets transformation process. It uses Virtuoso RDF store as a back-end for storing initial datasets and outputting the final, transformed datasets. Also, it is uses PostgreSQL/PostGIS for spatial indexing and processing, where spatial features are stored in an attribute of a PostgreSQL table of special type `GEOMETRY` or `GEOGRAPHY`. PostGIS supports all objects and functions in the OGC Simple Features for SQL specification [OGC10], R-tree-over-GiST (Generalised Search Tree) [HNP95] spatial indexes are employed in the evaluation of spatial queries. Also, PostGIS utilizes algorithms for indexing selectivity in order to provide high performance query plans for queries with mixed (spatial and non-spatial) predicates.

Moreover, FAGI-gis utilizes the external libraries `Apache Jena`, and `Google Guava`, that have been described in 3.1.1.1.
3.2.1.2 Components

FAGI-gis consists of four basic components, as depicted in Figure 6: The **GUI component** that provides the interface to the user to access the rest of the components, the **POSTGIS component** that handles the geospatial indexing and processing, the **CORE component** that handles the distinction of spatial and non-spatial properties, their loading into the database, scoring and fusing and the **TRANSFORMATIONS component** that implements the supported fusion actions through geometry transformations. Next, we describe in more detail the functionality of each component.

- **GUI component.** It consists of three interfaces, implementing a two step transformation process. The first two interfaces allow the user to configure the PostgreSQL/PostGIS database connection and to load the datasets to be process, providing information about the respective SPARQL endpoints and graph URIs. The third interface handles the actual fusion process, allowing the user to score the matching between geometries of the interlinked entities, to select a group of entities for fusion and, finally, fuse them, selecting one of the available fusion actions.

- **POSTGIS component.** This component handles the construction and initialization of a new PostGIS database. Also, it provides methods for importing RDF triples to the PostGIS DB: it establishes connection to a specified database, it loads endpoint/graph information to the appropriate dataset info table, it loads conventional and spatial RDF properties to the respective tables, etc.

- **CORE component.** This component provides the infrastructure for exporting metadata and geometric triples from a dataset with a given SPARQL endpoint and loading them into a PostGIS DB. Also, it provides methods for obtaining RDF links, scoring and then applying fusion transformations against them.

- **TRANSFORMATIONS component.** This component provides functionality for performing fusion transformations. To this end, it implements functions for fusing geometries of given nodes, assessing transformations on their suitability for fusing a specific pair of geometries and writing back fused geometries. Currently, it supports the seven transformations that are given below, while new transformations are being added as part of ongoing work on the tool:
  - Fuses two point geometries by averaging their x,y values.
  - Keeps both given geometries
  - Keeps source dataset geometry
  - Keeps target dataset geometry
  - Keeps the geometry with the most points (more complex geometry) and translates it so that its centroid matches the centroid of the other geometry
  - Keeps the geometry with the most points (more complex geometry)
  - Parametric transformation that keeps one of the two geometries and scales it. The geometry to keep and the scale factor are configurable.
In Section 6.2 of the Appendix, a more detailed description of the basic functionality of each component's class is provided.
3.2.1.3 Supported data sources and formats

Currently, FAGI-gis supports importing data from a Virtuoso SPARQL endpoint where the actual endpoint and the graph URI of the dataset are required. The supported RDF triples format is N-triples. The output of the tool is written into the underlying Virtuoso RDF store. Also, the currently supported vocabulary for representing geospatial features is GeoSPARQL with WKT serialization of geospatial features. Implementation of support of RDF files as input/output sources and support of more vocabularies are part of ongoing work.

3.2.2 Manual

3.2.2.1 Installation/Building instructions

This initial version of FAGI-gis is publicly available, offering the entire Java source code as well as a JAR package that contains executable binaries [FAGI-gis]. The JAR package can be run directly, given than Java JRE (or SDK) 1.7 (or later) are installed and a Virtuoso store and a PostgreSQL/PostGIS database are set up on the machine where the tool runs. Specifically, we used Virtuoso v07.00.3203 and PostgreSQL v9.19 w/ PostGIS v2. FAGI-gis has been successfully tested in both MS Windows and Linux environments. The tool’s code is organized in a Maven Project that can be directly loaded into an IDE (NetBeans) and be further developed.

3.2.2.2 Usage - Demonstration scenario

Next, we demonstrate the usage of the tool through the graphical user interface. In the first pane (Figure 7) the user is required to input connection information for the PostgreSQL database. Next (Figure 8), for both source and target dataset, the SPARQL endpoint and the graph URI of the dataset are required, similarly to FAGI-tr.

![Figure 7: Postgres DB configuration](image)

After the input datasets information is provided, the user selects to import the data, that is load and index the datasets in PostgreSQL/PostGIS. Through this process spatial and non-spatial properties and values are separately handled and stored into the database tables, so that spatial features are indexed and handled more efficiently. The separation of the properties (metadata) is performed by two SPARQL query restrictions:
Non-spatial metadata

  FILTER (regex(?s, "<subjectRegex>", "i"))
  FILTER (!regex(?p, "http://www.opengis.net/ont/geosparql#hasGeometry", "i"))
}

and

Spatial metadata

{ _:a ?p1 :g.
  _:a ?p2 ?g.
  FILTER(regex(?s, "<subjectRegex>", "i"))
  FILTER(regex(?p1, "http://www.opengis.net/ont/geosparql#hasGeometry", "i"))
  FILTER(regex(?p2, "http://www.opengis.net/ont/geosparql#asWKT", "i"))
}

filtering conventional and spatial triples respectively. In order to properly store the above information, the tools creates in PostgreSQL the tables described in Section 6.3 of Appendix.

Figure 8: Dataset loading and geospatial features indexing

Currently, the supported format for importing RDF metadata into PostgreSQL, through FAGI-gis is the GeoSPARQL standard with WKT serialization of geospatial features. However, by enriching the described restrictions, it is straightforward to support any other vocabulary, as we intend to do in the next version of the tool.
Finally, in the fusion pane, depicted in Figure 9, the user loads the links of the interlinked entities and, first, selects to score the similarity of the entities to be fused. In this current version, the scoring function simply examines whether, w.r.t. the selected fusion action, it is meaningful to fuse the geometries of two interlinked entities. As ongoing work on the tool, we are implementing geometry similarity functions (based on the facilities provided by PostGIS) that we intend to combine with similarity scores of conventional entity properties, in order to produce aggregated similarity scores of the entities to be fused. Also, we intend to provide to the user the option to select and combine individual similarity functions, that will help the user decide whether to fuse two entities, and if so, with which fusion strategy, or to discard the link between the two entities.

Finally, after the scoring is performed, the user can select a group of entity pairs and apply a fusion action of her selection. Currently, we have implemented six fusion actions:

- keep target,
- keep source,
- keep both,
- keep the geometry containing the most points (most complex geometry),
- keep the geometry containing the most points (most complex geometry) and shift it so that it has as centroid the centroid of the other geometry
- keep the average of two points
- keep one of two geometries, scaling it to some selected factor

As part of our ongoing work, we are enriching the set of available transformation functions, so that we cover several fusion scenarios/strategies, as described in Section 2.2. The fused geometries are first output in PostgreSQL tables and eventually written as triples into Virtuoso.
3.2.3 Licensing

The FAGI-gis tool is a free software and its current version (including the Java source code and sample data) is available from [FAGI-gis]. 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.

3.2.4 Future work

Similarly to FAGI-tr our short-term priorities for extending FAGI-gis involve adding (a) support of several RDF vocabularies and triple formats and (b) a more refined filtering mechanism for loading triples, that adhere to specific restrictions expressed in SPARQL.

Another direction is the enhancement of the fusion similarity function; the currently implemented function outputs a boolean value that denotes whether the two geometries can be fused, w.r.t. a specific fusion action. As a future step we intend to support several similarity functions both on conventional RDF triple values (e.g. string similarity functions on textual properties) and on geospatial triples (e.g. Hausdorff, Centroid, Symmetric Distances, etc.). A further functionality will allow the user to combine such functions on different datasets or even different sets of linked entities within two datasets. Finally, new, PostGIS-based functions are currently being added to support more fusion actions.

In our mid-term plans for the tools, we include the following tasks: (a) adding property mapping functionality, so that non-spatial properties can be matched and participate in the fusion process, (b) adding functionality for link rejection based on scoring the similarity of linked entities and (c) adding a neighbour searching functionality for discovering unlinked entities and adding new links or new entities to the respective datasets (as described in Sections 2.2 and 2.3).
4. Conclusion

In this deliverable, we defined the integrated framework for fusing geospatial RDF data that will be implemented through GeoKnow project. First, we provided a short description of the data fusion problem in general, as well as of the individual tasks of geospatial data fusion and RDF data fusion. Next, we presented a series of challenges w.r.t. fusing geospatial RDF data and introduced the envisioned framework for Fusion and Aggregation for Geospatial Information - FAGI. Finally, we presented the first two tools of the framework, FAGI-tr and FAGI-gis, that focus on fusing different representations of geometry in RDF.

Our next steps involve: (a) enhancing the functionality of the implemented tools, as described through Chapter 3, (b) implementing property mapping functionality so that we can fuse non-spatial metadata and also exploit them to fuse the spatial features of the respective entities and (c) implementing an initial learning functionality for producing (semi)automatic fusion recommendations.
5. References


[FAGI-tr] FAGI-tr. Available at https://github.com/GeoKnow/FAGI-tr


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


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


# 6. Appendix

## 6.1 Basic FAGI-tr classes

### CORE

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GeometryFetcher</td>
<td>responsible for fetching geometry related triples from a source dataset</td>
</tr>
<tr>
<td>Matcher</td>
<td>responsible for matching rules against a given dataset and providing metadata relevant to the rule.</td>
</tr>
<tr>
<td>MetadataFetcher</td>
<td>responsible for fetching all metadata triples (non geometry related) from a source dataset and copying</td>
</tr>
<tr>
<td></td>
<td>them first to the specified by the dataset local unmodified graph and second, if the populateTransformationGraph flag is set, to the local transformed graph</td>
</tr>
</tbody>
</table>

### GEOMETRY

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GeometryProcessor</td>
<td>Container class for providing geometry processing functionality</td>
</tr>
<tr>
<td>EastingNorthingProcessor</td>
<td>Processor for Easting/Northing type geometry. extends GeometryProcessor. creates EastingNorthingCRSParser, geometryParser, geometryWriter, geometryCRS</td>
</tr>
<tr>
<td>GMLProcessor</td>
<td>Processor for Geometry Markup Language (GML) format geometry. extends GeometryProcessor</td>
</tr>
<tr>
<td>LatLongProcessor</td>
<td>Processor for Lat/Long format geometry. extends GeometryProcessor</td>
</tr>
<tr>
<td>WKTProcessor</td>
<td>Processor for WKT format geometry. extends GeometryProcessor</td>
</tr>
</tbody>
</table>

### GEOMETRY.CRS

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AbstractCRSParser</td>
<td>Abstract class providing CRS parsing functionality.</td>
</tr>
<tr>
<td>CRSTransformer</td>
<td>Provides functionality for transforming geometry from one CRS to another.</td>
</tr>
<tr>
<td>EastingNorthingCRSParser</td>
<td>CRS parser for easting/northing type geometry. extends AbstractCRSParser</td>
</tr>
<tr>
<td>GMLCRSParser</td>
<td>CRS parser for GML type geometry. extends AbstractCRSParser</td>
</tr>
<tr>
<td>GeometryCRS</td>
<td>CRS SRID, authority</td>
</tr>
<tr>
<td></td>
<td>Representation for a geometry CRS. Constructor for a CRSSs (returns GeometryCRS object)</td>
</tr>
<tr>
<td>LatLongCRSParser</td>
<td>CRS parser for lat/long type geometry. extends AbstractCRSParser. (Returns GeometryCRS object)</td>
</tr>
<tr>
<td>WKTCRSParser</td>
<td>CRS parser for WKT type geometry. (Returns GeometryCRS object)</td>
</tr>
</tbody>
</table>

### GEOMETRY PARSERS

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GeometryParser</td>
<td>Interface for parsing geometry to internal representation.</td>
</tr>
</tbody>
</table>
**EastingsNorthingParser**
Geometry parser for easting/northing type geometry. implements GeometryParser

**GMLParser**
Geometry parser for GML type geometry. implements GeometryParser

**LatLongParser**
Geometry parser for lat/long type geometry. implements GeometryParser

**WKTParser**
Geometry parser for WKT type geometry. implements GeometryParser

---

**GEOMETRY WRITERS**

**GeometryWriter**
Interface for writing geometry serialization from internal representation.

**EastingsNorthingWriter**
Geometry writer for easting/northing type geometry. implements GeometryWriter

**GMLWriter**
Geometry writer for GML type geometry. implements GeometryWriter

**LatLongWriter**
Geometry writer for lat/long type geometry. implements GeometryWriter

**WKTParser**
Geometry writer for WKT type geometry. implements GeometryWriter

---

**GEOMETRY EXCEPTIONS**

**CRSParseException**
CRSParseException signifies a failure when parsing the Coordinate Reference System from the serialization of a geometry.

**GeometryParseException**
GeometryParseException signifies a failure when parsing the serialization of a geometry into a Geometry object.

**GeometrySerialisationException**
GeometrySerialisationException signifies a failure when attempting to serialize a Geometry object.

---

**GUI**

**FetcherGUI**
Application entrypoint. extends javax.swing.JFrame implements MessageListener

**DatasetPanel**
Dataset Panel

**FetcherTransformerPanel**
Fetcher/transformer panel. Implements RuleListener, DatasetListener, FetcherListener

**MatcherPanel**
Matcher Panel + Gets rules from config files, implements DatasetListener

---

**GUI LISTENERS**

**DatasetListener**
Listener for dataset information - for FetcherTransformerPanel and MatcherPanel implementation
### FETCHER INTERFACE

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FetcherListener</td>
<td>Fetcher exception message listener - for TransformerPanel implementation Interfaces</td>
</tr>
<tr>
<td>MessageListener</td>
<td>Message listener - for FetcherGUI implementation</td>
</tr>
<tr>
<td>RuleListener</td>
<td>Matched rule listener - for TransformerPanel implementation</td>
</tr>
</tbody>
</table>

### GUI WORKERS

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBConnectionParameters</td>
<td>Container object holding database connection parameters. (url, username, password)</td>
</tr>
<tr>
<td>Dataset</td>
<td>An object that contains information about a given dataset.</td>
</tr>
<tr>
<td>FetcherWorker</td>
<td>This worker handles fetching metadata and geometry triples from specified datasets and transforming geometries if requested. Creates GeometryFetcher from core package</td>
</tr>
<tr>
<td>MatchedRule</td>
<td>Represents a rule that has been matched against a dataset.</td>
</tr>
<tr>
<td>MatcherWorker</td>
<td>This worker handles matching a set of rules against a dataset. Calls Matcher from core pckg</td>
</tr>
<tr>
<td>RuleConfig</td>
<td>Container object holding locations of rule configuration files.</td>
</tr>
</tbody>
</table>

### RULES

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rule</td>
<td>Represents a rule for matching triples in a dataset.</td>
</tr>
<tr>
<td>RulePattern</td>
<td>Represents a rule pattern for matching triples in a dataset. Has a copy constructor existing rule pattern to be copied</td>
</tr>
<tr>
<td>RuleTriple</td>
<td>Represents an unparsed rule for matching triples against a dataset as it is read from the configuration files. Declaring variable types representing variables with WKT, GML, latitude, longitude, etc.. geometry literals</td>
</tr>
<tr>
<td>Variable</td>
<td>Represents a variable with its name, its geometric type and optional its value (content) and datatype.</td>
</tr>
</tbody>
</table>

### RULES PARSER

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RuleConfigParser</td>
<td>Parser for rule configuration files</td>
</tr>
<tr>
<td>RulePatternParser</td>
<td>Rule pattern parser.</td>
</tr>
<tr>
<td>RuleQueryUtils</td>
<td>Utilities for forming queries from rules</td>
</tr>
</tbody>
</table>

### 6.2 Basic FAGI-gis classes

### CORE

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Importer</td>
<td>Provides the infrastructure for the export of metadata and geometric triples from a dataset with a given SPARQL endpoint and their loading into a PostGIS db using an instance of PostGISImporter.</td>
</tr>
</tbody>
</table>
Methods:
- **importMetadata** → Fetches all triples with a subject matching parameter `subjectRegex` and a predicate not matching `Importer#HAS_GEOMETRY_REGEX`. Those triples are then imported into a PostGIS database using an instance of `PostGISImporter`.
- **importGeometries** → This method fetches all triples with a subject matching parameter `subjectRegex` and a two triple chain with a predicate matching `Importer#HAS_GEOMETRY_REGEX` in the first triple and `Importer#AS_WKT_REGEX` in the second triple.

**GeometryFuser**
Provides methods for obtaining RDF links, scoring and then applying fusion transformations against them.

Methods:
- **fuse** → Apply given fusion transformation on list of links.
- **score** → Scores given fusion transformation for each link in list.
- **parseLInksFile** → Parses given RDF link file.
- **connect** → Connects to the database

**Link**
Represents a link between two RDF nodes.

### GUI

**DatasetPanel**
Handles setting of database configuration parameters.

**FuserPanel**
Handles application of fusion transformations. Implements `DBConfigListener`.

**FusionGISGUI**
Application entry point. Implements `ErrorListener`.

**ImporterPanel**
Handles importing of RDF graph into a PostGIS db.

**ScaleParamsPanel**
Handles setting of parameters for gr.athenainnovation.imis.fusion.gis.transformations.ScaleTransformation.

### GUI LISTENERS (interfaces)

- **DBConfigListener**
  Listener for changes to the database configuration. Interface for `FuserPanel`

- **ErrorListener**
  Error message listener. Interface for `FusionGISGUI`

### GUI WORKERS

- **DBConfig**
  Stores connection data for a database (db name, username, password).

- **Dataset**
  Stores information about a dataset accessible via SPARQL endpoint (endpoint, graph, subjectRegex).

- **FuseWorker**
  This worker handles application of a transformation against a set of links.

- **ImporterWorker**
  Exports triples from a dataset using its SPARQL endpoint and then imports them into a PostGIS database. Publishes progress.

- **ScoreWorker**
  This worker handles scoring a transformation against a set of links.

### POSTGIS

- **DatabaseInitialiser**
  Handles construction and initialization of a new PostGIS database.
PostGISImporter: The class provides methods for importing RDF triples to the PostGIS DB (establishes connection to specified db, loads endpoint/graph to the appropriate dataset info table, loads an RDF statement to the appropriate dataset metadata table, loads geometric information to the appropriate dataset geometries table, closes connection).

ScriptRunner: Handles parsing of sql scripts from file and executing them on a database.

TRANSFORMATIONS

AbstractFusionTransformation: Abstract class for the definition of fusion transformations. (methods for: fusing geometries of given nodes, scoring transformation on its suitability for fusing the geometries of given nodes, inserting fused geometry into database.)

AvgTwoPointsTransformation: Fuses two point geometries by averaging their x, y values.

KeepBothTransformation: Keeps both given geometries.

KeepLeftTransformation: Keeps left (nodeA) geometry.

KeepMostPointsAndTranslateTransformation: Keeps the geometry with the most points and translates it so that its centroid matches the centroid of the other geometry.

KeepMostPointsTransformation: Keeps the geometry with the most points.

KeepRightTransformation: Keeps right (nodeB) geometry.

ScaleTransformation: Parametric transformation that keeps one of the two geometries and scales it. Both which geometry is kept and the scale factor are configurable.

6.3 Postgres DB schema used by FAGI-gis

--Drop all tables if they exist

DROP TABLE IF EXISTS dataset_a_info;
DROP TABLE IF EXISTS dataset_a_metadata;
DROP TABLE IF EXISTS dataset_a_geometries;
DROP TABLE IF EXISTS dataset_b_info;
DROP TABLE IF EXISTS dataset_b_metadata;
DROP TABLE IF EXISTS dataset_b_geometries;
DROP TABLE IF EXISTS fused_geometries;

--Create a table to hold dataSetA's info

CREATE TABLE dataset_a_info (endpoint text NOT NULL, graph text NOT NULL);

--Create a table to hold dataSetA's metadata

CREATE TABLE dataset_a_metadata (  
id serial PRIMARY KEY,  
subject text NOT NULL,  
predicate text NOT NULL,  
object text NOT NULL,  
object_lang text,  
object_datatype text  
);  
CREATE INDEX idx_dataset_a_metadata_subject ON dataset_a_metadata USING btree (subject);

--Create a table to hold datasetA's geometries  
CREATE TABLE dataset_a_geometries (  
id serial PRIMARY KEY,  
subject text NOT NULL,  
predicate text NOT NULL,  
object text NOT NULL,  
object_lang text,  
object_datatype text  
);  
SELECT AddGeometryColumn('dataset_a_geometries', 'geom', 4326, 'GEOMETRY', 2);  
CREATE INDEX idx_dataset_a_geometries_geom ON dataset_a_geometries USING gist (geom);  
CLUSTER dataset_a_geometries USING idx_dataset_a_geometries_geom;

--Create a table to hold datasetB's info  
CREATE TABLE dataset_b_info (  
endpoint text NOT NULL,  
graph text NOT NULL  
);

--Create a table to hold datasetB's metadata  
CREATE TABLE dataset_b_metadata (  
id serial PRIMARY KEY,  
subject text NOT NULL,  
predicate text NOT NULL,  
object text NOT NULL,  
object_lang text,  
object_datatype text  
);  
CREATE INDEX idx_dataset_b_metadata_subject ON dataset_b_metadata USING btree (subject);

--Create a table to hold datasetB's geometries  
CREATE TABLE dataset_b_geometries (  
id serial PRIMARY KEY,  
subject text NOT NULL,  
predicate text NOT NULL,  
object text NOT NULL,  
object_lang text,  
object_datatype text  
);  
SELECT AddGeometryColumn('dataset_b_geometries', 'geom', 4326, 'GEOMETRY', 2);  
CREATE INDEX idx_dataset_b_geometries_geom ON dataset_b_geometries USING gist (geom);  
CLUSTER dataset_b_geometries USING idx_dataset_b_geometries_geom;

--Create a table to hold fused geometries  
CREATE TABLE fused_geometries (  
...
id serial PRIMARY KEY,
subject_A text NOT NULL,
subject_B text NOT NULL
);
SELECT AddGeometryColumn('fused_geometries', 'geom', 4326, 'GEOMETRY', 2);
CREATE INDEX idx_fused_geometries_geom ON fused_geometries USING gist (geom);