REPORT ON DELIVERABLE D3.3

Initial release of the fusion framework

SLIPO

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

Horizon 2020

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

The information in this document reflects 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.

Dissemination Level	Public
Due Date of Deliverable	Month 18, 30/06/2018
Actual Submission Date	21/06/2018
Work Package	WP3 - Data Integration and Quality Assurance
Tasks	3.3 Fusion
Type	Other
Approval Status	Submitted for approval
Version	1.0
Number of Pages	75
Filename	D3.3_Initial_release_of_the_fusion_framework.pdf

Abstract

This document presents the beta version of FAGI, our software framework for POI fusion. First, we briefly describe the current state of the art in geospatial data fusion, as well as the initial version of the FAGI software, which comprised the starting point of our work. Then, we present FAGI v2.0, the current version optimized for the scalable and quality assured fusion of POIs. Apart from implementation and deployment information, we thoroughly describe the POI-specific algorithms, similarity functions, and fusion rules/actions we developed and tested against real-world, commercial POI datasets. Finally, we present our evaluation experiments assessing both the fusion effectiveness and the scalability of the software.

History

version	date	reason	revised by
0.1	30/04/2018	Deliverable structure and initial content	Giorgos Giannopoulos
0.2	10/05/2018	Added content on FAGI components and architecture section	Nikos Karagiannakis
0.3	15/05/2018	First draft	Giorgos Giannopoulos, Nikos Karagiannakis
0.4	18/05/2018	Multiple revisions and improvements	Spiros Athanasiou
0.5	13/06/2018	Multiple revisions and improvements	Giorgos Giannopoulos, Jens Lehmann, Fabrizio Orlandi
0.6	18/05/2018	Multiple revisions and improvements	Spiros Athanasiou, Dimitrios Skoutas
0.7	14/06/2018	Multiple revisions and improvements	Giorgos Giannopoulos, Dimitrios Skoutas
0.8	20/06/2018	Reviewed version	Mohamed Sherif
1.0	30/06/2018	Final version	Spiros Athanasiou

Author list

organization	name	contact information
ATHENA RC	Spiros Athanasiou	spathan@imis.athena-innovation.gr
ATHENA RC	Giorgos Giannopoulos	giann@imis.athena-innovation.gr
ATHENA RC	Nikos Karagiannakis	nkaragiannakis@imis.athena-innovation.gr
ATHENA RC	Dimitrios Skoutas	dskoutas@imis.athena-innovation.gr
IAIS	Jens Lehmann	lehmann@informatik.uni-leipzig.de
IAIS	Fabrizio Orlandi	Fabrizio.Orlandi@iais.fraunhofer.de

Executive Summary

This document presents the beta version of FAGI, our software framework for the scalable and quality assured fusion of Point of Interest (POI) datasets. Fusion consists in receiving two datasets containing POIs and their properties (attributes), as well as a set of links linking POI entities between the two datasets, and producing a third, final dataset, which contains consolidated descriptions of the linked POIs. Each POI entity in the final, fused dataset is described by a set of non-redundant, non-conflicting, complete, properties, that have been derived by merging the initial descriptions for the linked POIs. FAGI implements a series of rules and actions for link validation and fusion, as well as mechanisms for selecting the proper fusion actions in each case. Further, FAGI incorporates mechanisms for fusion validation and quality statistics/indicators extraction to facilitate and assure the quality of the fusion process.

The presented version of FAGI extends the initial framework (FAGI v1.0), a map-based, user-interactive platform for manually performing property matching and fusion actions on individual properties of linked geospatial entities. The current version, FAGI v2.0, developed in the context of SLIPO, has been extended to specialize on the effective and scalable fusion of POI entities. To this end, in order to implement the core mechanisms for POI fusion (*property similarity functions, training features, fusion conditions, fusion rules and fusion actions*), we first performed an analysis of real world, commercial POI datasets. The analysis engaged the industrial partners of the project, which provided annotations (*training labels*) on samples of the datasets, w.r.t. fusion and validation actions. This analysis and iterative experimentation allowed us to fine-tune the fusion and validation mechanisms on real world data. Further, it facilitated the development of initial learning mechanisms that allow the automatic validation and selection of fusion actions.

The layout of document is the following.

In Section 1, we introduce the setting of the fusion task. We first describe the objectives of fusion in the frame of the project, and then provide some background knowledge, briefly presenting existing works on the fusion of geospatial entities. Finally, we briefly report our achievements until M18 of the project and the delivery of FAGI v2.0.

In Section 2, we revisit the roadmap for the development of FAGI, present the evolution of the software from its initial version (v1.0) at the start of the project, and we summarize the major advancements and key features of FAGI v2.0 achieved during the first period of the project.

In Section 3, we present our work on developing POI-specific similarity functions and learning mechanisms, towards facilitating and automating the link validation and fusion processes within FAGI.

In Section 4, we present in detail the innerworkings of FAGI. Specifically, we describe all implemented mechanisms that allow the design and execution of fusion specifications, and the extraction of quality indicators on the fusion process. Further, we present the architecture and main components of FAGI.

In Section 5, we present a user guide for FAGI v2.0, including building and installation instructions, configuration settings, and a short demonstration of the usage of the software.

Finally, in Section 6 we evaluate the scalability of FAGI against large commercial POI datasets.

Abbreviations and Acronyms

API	Application Programming Interface
FAGI	Fusion and Aggregation of Geospatial Information
CRS	Coordinate Reference System
KPI	Key Performance Indicator
ML	Machine Learning
POI	Point of Interest
RDF	Resource Description Framework
SPARQL	SPARQL Protocol and RDF Query Language
UI	User Interface
VM	Virtual Machine
WGS84	World Geodetic System 1984 (EPSG:4326)
XML	Extensible Markup Language

Table of Contents

1. Introduction 9

1.1. POI data fusion 9

1.1.1. Fusion task definition 9

1.1.2. Fusion importance and challenges 10

1.2. State of the art 11

1.2.1. Existing approaches 11

1.2.2. Challenges and shortcomings 13

1.3. Fusion in the SLIPO lifecycle 14

1.3.1. Achievements - FAGI v2.0 15

2. The FAGI Framework 17

2.1. FAGI v1.0 17

2.2. Towards FAGI v3.0 19

2.3. Current Features and Functionality 21

3. Optimizing fusion of POIs 24

3.1. Analysis of linked POI datasets 24

3.1.1. Problem formulation 24

3.1.2. Analysis process 25

3.1.3. Analysis outcomes 27

3.2. Similarity-based link validation 29

3.2.1. FAGISimilarity 29
3.2.2. Supervised link validation and fusion 33
4. FAGI v2.0 39
4.1. Major features and functionality 39
4.1.1. POI fusion 39
4.1.2. POI link Validation 44
4.1.3. Recommending POI link validation and fusion 47
4.1.4. Quality indicators and statistics 49
4.1.5. Dataset output modes 50
4.2. Architecture 51
4.2.1. Input and output 52
4.2.2. Core modules 53
4.2.3. Partitioning scheme 54
4.3. Libraries and Frameworks 55
4.4. License 57
4.5. Documentation 57
5. Usage Manual 58
5.1. Building Installation 58
5.1.1. Command line version 58
5.1.2. Web interface version 58
5.2. Configuration Settings 59

5.2.1. Configuration specification 59
5.2.2. Rule Specification 62
5.3. Demonstration 63
5.3.1. FAGI input configuration 63
5.3.2. FAGI validation and fusion rule specification 64
5.3.3. FAGI execution 68
6. Experimental Evaluation 69
6.1. Datasets and measures 69
6.1.1. Datasets 69
6.1.2. Measures 70
6.2. Results 70
7. References 72
8. Annex 74
8.1. Datasets characteristics 74
8.1.1. Dataset A 74
8.1.2. Dataset B 74

1. Introduction

In this section, we first discuss the basic concepts and goals of POI data fusion. Then, we present the current state of the art in POI fusion approaches and frameworks, and finally, we briefly present our goals and achievements regarding the scalable, effective, and quality-assured fusion of POIs.

1.1. POI data fusion

1.1.1. Fusion task 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**.

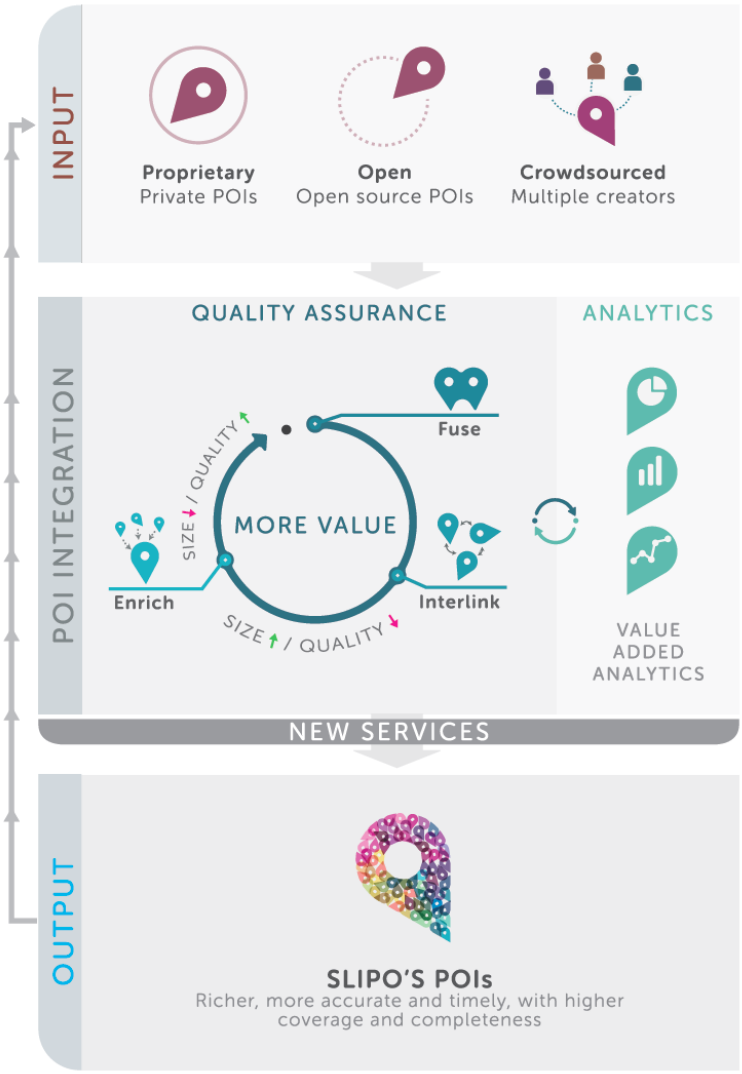


Figure 1: POI integration lifecycle

Fusion constitutes the final part of the data integration process, which usually consists of three steps: schema integration (mapping), duplicate detection (interlinking) and fusion [BN08]. In the frame of the SLIPO project, fusion focuses on Point of Interest (POI) entities, that are characterized by a set of major properties (*name, coordinates, category*), as well as potentially several additional properties (*address, telephone, email, rating, etc.*). Fusion considers two input datasets containing POIs and their properties and a set of links that connect POI entities between the two datasets. Mapping between the properties of the linked entities and interlinking of the POI entities between the two datasets are performed in previous steps of the SLIPO integration workflow (*see TripleGeo [SLIPOD22] and LIMES [SLIPOD31] software respectively*). The goal of fusion is to produce a final dataset, which contains consolidated descriptions of the linked POIs. That is, each POI entity in the final, fused dataset must be described by a set of **non-redundant, non-conflicting, complete properties, that have been derived by merging the initial descriptions for the POI**. Additionally, considering the big picture of the POI integration lifecycle (Figure 1), the fusion process is tightly interconnected with validation and quality assurance. To this end, the fusion process needs to incorporate several mechanisms to assess the quality of the proposed fusion actions and their results.

1.1.2. Fusion importance and challenges

The fusion process starts at the point where a set of pairs of linked POI entities is created, and consequently, each pair carries along several duplicate descriptions (values) regarding each of their properties. For example, a pair of linked POIs potentially comprises two name properties, two coordinate fields and two addresses. The values of these pairs of matching properties might be exactly equal; in that case, fusing the values of the individual properties is straightforward. However, this is usually not the case; the values of the matching properties differ due to various reasons (*e.g., formatting, spelling errors, measurement-curation-validation errors, multiple valid attributes for the same real-world characteristic*) and to various extents (*from a simple syntax error in a name, to completely different phone numbers*).

Understandably, this setting may significantly hinder the qualitative assessment of both the initial POI data, and the fusion results. For example, if two linked POI entities, with very high *geospatial proximity*, have moderately different names, should they be validated as the same real-world POI, or should their link be rejected as wrong? In this case, the decision usually depends on the semantics of the POIs (*e.g., which POI categories they belong to*), on the semantics of specific terms that comprise their names (*e.g., is the POI a neighborhood of a city or a gas station?*), on the combination of similarities on several individual matching properties of the POIs, etc. On the other hand, POI descriptions are often *incomplete*, lacking even major property values, such as address street or number, phone and even POI category. In this case, the user has even less information available to decide whether and how two POIs should be fused. In another example, the user/system may be very confident that the pair of POIs correspond to the same real-world POI (*e.g., most of their matching properties have the same or very close value*). However, only the names of the two POIs are moderately different. In that case, the user has several options as to how to fuse the names of the POI: *select one of the two values*; *keep both values in separate properties*; *merge (i.e., concatenate) the two values and keep them in the same property*; *keep the most recent value*; *keep the value from the most trustworthy dataset*, etc. Again, deciding the most proper option differs from case to case and depends on several factors.

It becomes apparent that several Key Performance Indicators (KPIs) of the POI integration lifecycle directly depend on the effective application of fusion mechanisms: Accuracy of POI location; Accuracy of POI categories; Completeness; Timeliness; Lack of conflicting POIs and/or properties (*see D1.1 "Use Cases and Requirements", Sections 2.1.3.15-2.1.3.18 [SLIPOD11]*). One of the most important contributions of fusion is increasing the quality of a dataset, with respect to increasing the accuracy of values and handling conflicting values. Additionally, increasing the completeness and improving the timeliness of the data are important goals towards increasing the value of POI datasets. Thus, fusion comprises a crucial step of the integration process that increases the value of POI data by consolidating them and improving their quality.

1.2. State of the art

1.2.1. Existing approaches

Traditionally, geospatial fusion (or conflation) consists in merging different geospatial datasets of overlapping regions, in such way that the best quality elements of individual datasets are kept in the final dataset [CK08]. Reviewing the considerable amount of approaches, some standardized techniques have emerged, which typically operate in three main steps. *Feature matching* aims at finding a set of conjugate point pairs (control points) in two datasets which most probably correspond to each other and can be used as reference points. These are usually identified based on strict geospatial matching. Then, *match checking* increases the accuracy of control points by filtering out pairs with ambiguous quality. This can also utilize non-spatial metadata of objects to produce set of more accurate control points. Finally, *spatial attribute alignment* uses the control points to align the rest of the geospatial objects in both datasets by space partitioning and transformation techniques. Such methods utilize a combination of spatial and non-spatial criteria in matching/scoring functions [Saa88][CCM+98], statistics on geospatial properties/relations [WF99][CSK+06], or iteration [Saa88].

In the last years, the focus of geospatial data fusion tasks has shifted from traditional geometry conflation to **non-geospatial (*thematic*) metadata fusion**. This can be partially attributed to the advancements in mapping and geocoding that resulted in the production of more accurate geospatial data (*in terms of POI geometries and coordinates*). On top of that, being able to provide a set of very accurate thematic metadata for POIs became the number-one priority for commercial applications. For example, it is currently more important to provide a correct and as specific as possible *category* for a POI (*e.g., "Mexican fast food court", instead of just "restaurant"*) or up-to-date information (*e.g., a cafeteria that has closed and a cocktail bar has opened at the exact same spot*), rather than provide the precise geometry of a POI. If we also consider the rise of crowdsourcing, social networks, and the adoption of Linked Data practices, we find ourselves in an era where geospatial information is accompanied by *a wealth of non-spatial metadata* (*e.g., textual descriptions, categories/tags, reviews, ratings*). Thus, previous fusion approaches might be considered to a large extent as *obsolete*, at least when considering their objective in the context of POI integration: the major goal of fusion is not so much to align maps, but rather to decide on which *fusion strategies* (*e.g., keep both, keep largest, keep more recent*) to be used in order to merge the properties that characterize a pair of linked POIs. In this context, several fusion frameworks have been recently developed to address the challenge of Linked Data fusion.

Sieve [MMB12] focuses on quality assessment and fusion of Linked Data, being part of a larger framework for Linked Data integration [SMI+12] that provides several techniques for data fusion. It takes into account factors such as timeliness of data, provenance, as well as user-configurable preference lists on features of the dataset. 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.

ODCleanStore [MK+12] is another framework that supports linking, cleaning, transformation and quality assessment operations on Linked Data. The fusion component supports several user configurable fusion strategies, which also consider provenance and quality metadata:

- 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

WInte.r [LBB17] is a recently released framework for end-to-end data integration. The framework implements well-known methods for data pre-processing, schema matching, identity resolution, data fusion, and result evaluation. Regarding fusion, the system expects as input datasets in a consolidated schema and correspondences (links) between their records. From these correspondences, groups of records which represent the same entity are collected for each attribute. Such an entity/attribute group then contains all values for this combination from the input datasets. To decide for a final value, a *conflict resolution function* (fusion action) is applied to the values. A selected *Data Fusion Strategy* defines the conflict resolution function and an *evaluation rule* for each attribute. These functions determine how a final value is chosen from multiple possible values and how it is evaluated. The supported options for fusion are:

- Voting: Applies majority voting to the values.

- ClusteredVote: Clusters all values using the provided similarity measure and returns the centroid of the largest resulting cluster.
- Intersection: Creates the intersection of all values (applicable if values are sets).
- IntersectionKSources: Creates a set of all values that are included in at least k input values (applicable if values are sets).
- Union: Creates the union of all values (applicable if values are sets).
- FavourSources: Returns the value from the source with the highest score (as defined by the user).
- MostRecent: Returns the value from the source that is most recent (as defined by the user).
- Average: Returns the average of all values (numeric).
- Median: Returns the median of all values (numeric).
- LongestString: Returns the longest value by character count (strings).
- ShortestString: Returns the shortest value by character count (strings).

FAGI v1.0 [GVK+15], the first version of our fusion software, is *the only available framework dedicated on fusion of geospatial Linked Data*. It supports an extended set of fusion actions regarding both geometries and thematic properties of the entities, as well as batch fusion actions and a basic link discovery functionality. Further, through its interactive map-based web interface, it supports authoring and quality assessment, allowing the collation of vector geometries of linked entities with the underlying map layers and the manual adjustment of geometries.

1.2.2. Challenges and shortcomings

Although a lot of effort has been put on research and development of schema matching and interlinking frameworks, only a handful of platforms handle the fusion part, as demonstrated in the previous sub-section. Further, most approaches on fusion usually reach the maturity level of a research prototype and then remain stagnant; as a result, there is a lack of widely established and adopted platforms for data fusion in general, and more specifically, for geospatial data fusion.

Another important issue is the fact that existing solutions on geospatial data integration mainly target on increasing the *efficiency* of the integration process [NH+15][PSC+16], neglecting the quality/accuracy aspect of integration. As such, most methods focus on developing optimized indexes, structures and schemes for performing an integration task (*e.g., interlinking*) very efficiently. The similarity functions they apply for comparing geospatial entities are generic, widely adopted functions that are not specialized, or even properly tuned for the setting of comparing geospatial entities. Moreover, most approaches on geospatial data integration mostly focus on toponyms, which is a more straightforward problem, rather than POI entities. However, as shown later in this document, POI entities present several specificities in their properties that need to be considered for engineering POI-specific similarity functions, rules, and thresholds for validation and fusion.

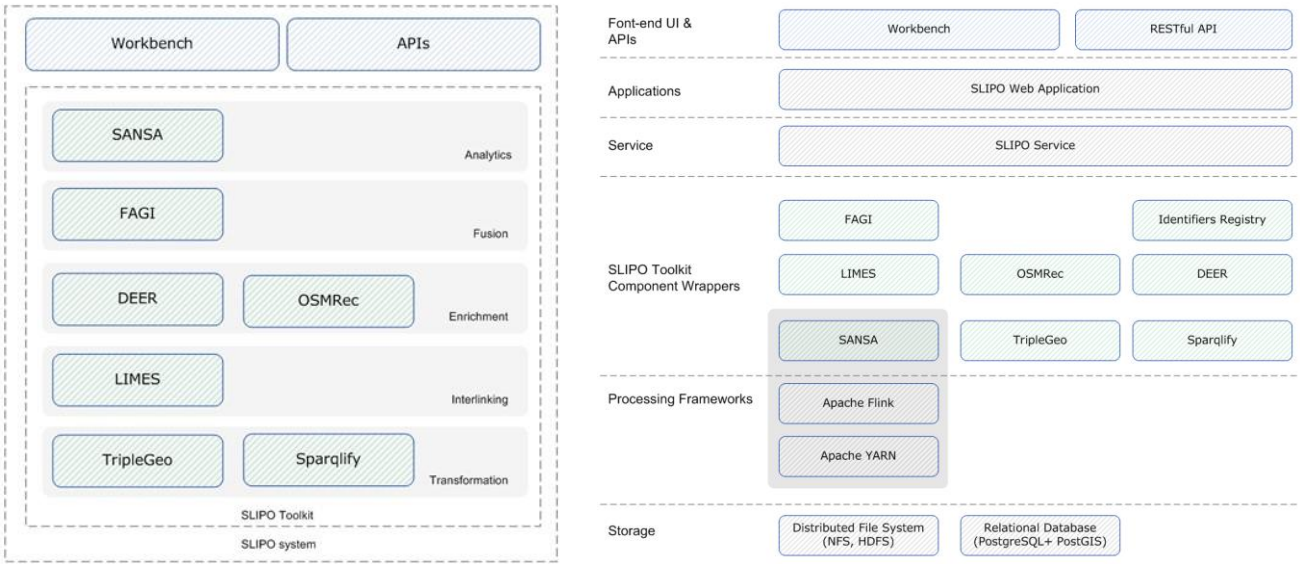
One of the most significant gaps in existing platforms for data fusion is the lack of *rule specification* or *machine learning* (ML) mechanisms for automatically deciding (*or recommending*): (a) whether to accept or reject a pair of linked entities (validation), and (b) which fusion action to perform on each pair of matching

properties of two linked entities. Every platform we have reviewed is limited to just providing a set of common fusion actions (*strategies*) and expecting from the user to select the appropriate one for each pair of matching properties. Some approaches support the manual creation of simple, static rules for deciding fusion actions depending on the types and/or values of selected properties. However, none of the approaches allows the construction of elaborate fusion rules that consider and combine information from several of the properties of linked entities, in order to select the most proper fusion action. Further, none applies ML techniques to learn validation and fusion models from training on historical data, which can then recommend validation and fusion actions for new sets of linked entities. A critical factor for the effectiveness of such algorithms is the proper selection of training features, i.e., features that describe pairs of linked entities and "relate" them to a validation/fusion action. Effectively implementing such methods would alleviate a significant bottleneck in the whole POI integration workflow: the one-by-one examination of each pair of linked/fused POIs to ascertain whether it was properly linked and fused. Of course, even a highly effective ML model is not expected to fully replace manual validation. However, it could as well render human effort unnecessary for a large proportion of the datasets, for which the rule specifications, or the ML model, outputs validation/fusion actions with very high confidence.

1.3. Fusion in the SLIPO lifecycle

The SLIPO POI integration lifecycle is realized through the SLIPO Workbench, a platform for defining, executing and managing POI integration workflows (*see Deliverable D1.3 "Beta SLIPO Integrated System" [SLIPOD13]*). These workflows combine all components of the SLIPO Toolkit, supporting the integrated execution of all four core POI integration steps: *transformation, interlinking, enrichment and fusion*. Additionally, the SLIPO system prescribes a set of value-added services on top of integrated POI datasets (Figure 2 (a)).

The goal of Task 3.3 is to deliver FAGI, the fusion framework of SLIPO. FAGI implements an extended functionality for performing automated fusion on the properties of linked POI entities, as well as quality indicators extraction and link validation. In the context of SLIPO, FAGI receives as input RDF POI datasets conforming to the SLIPO ontology. Thus, FAGI's input POI data are first transformed by TripleGeo, into the proper RDF format and schema. Further, apart from two input POI datasets, FAGI requires as input a file containing the links between the POIs of the two datasets, which is produced by LIMES. FAGI's main output consists in a single file, which contains consolidated POI entities from both input POI datasets.



(a) High level architecture

(b) Logical architecture

Figure 2: SLIPO Architecture

1.3.1. Achievements - FAGI v2.0

FAGI v2.0 is the next version of FAGI, developed in the context of the SLIPO project, and focuses on POI-specific fusion and validation. One of the major goals of the project is to abstract as much complexity as possible from the end users of the SLIPO Workbench. So, in order to keep user interaction at a minimum and require no knowledge of Linked Data technologies and concepts, we aimed at adapting and fine-tuning FAGI's functionality specifically for POI data, as well as at automating the fusion process as much as possible. To this end, we emphasized on the development of the backend of the platform, aiming to enrich and specialize the core fusion functionality of the framework. Next, we enumerate the new features and functionality of FAGI, achieved during the first 18 months of the project.

- **Specialization on POI data**. Our work was driven by the major commercial POI datasets that were available in the project: TomTom POIs; WIGeoGIS-Herold POIs; GET POIs. We explored and analyzed these datasets individually, but more importantly, against each other, with respect to the types of POIs they contain, and their naming conventions. Additionally, we explored several other properties of the POI entities (*name, address, phone, website, email*), examining the differences their values present in pairs of linked POIs. Finally, we collaborated with the industrial partners (*producers, owners or users of the commercial datasets*) in gathering training labels (*annotations*) on fusion and validation actions.

  Based on this analysis, we evaluated a series of **similarity measures**, extended and fine-tuned them, increasing their similarity matching accuracy on the POI datasets at hand. In parallel, utilizing the same similarity functions, we developed POI-specific **rule mechanisms** for deciding fusion actions, taking into account conditions on the property values of the linked POIs. On top of the developed functions, conditions, rules and actions, we have implemented a simple mechanism for constructing validation and fusion **rule specifications**, that allow the consideration of several factors for deciding a fusion action for a pair of matching properties or a validation action for a pair of linked POIs.

Finally, we created a first set of **predefined fusion rules** that are incorporated into FAGI and, apart from the core fusion functionality they offer, they serve as a guide for the end users of the platform into constructing new fusion rules, depending on the context, the datasets and the application scenario.

- **Quality-based fusion**. We implemented a dedicated component in FAGI v2.0 for extracting **statistics** and **quality indicators** from both the input datasets and the output (fusion) results. These involve statistics on the individual POIs within the input linked datasets, as well as statistics regarding the pairs of linked POIs. For example, they include distribution of specific POI properties, % of empty/non-empty property values, measures on how (dis)similar the values of matching properties for linked POIs are, etc.

  The purpose of this component is twofold: (a) present statistics and indicators to the end user that will assist her to *assess the quality, potential issues and gaps* in the input datasets; select the most *fitting fusion actions* per case; potentially re-configure and re-execute more appropriate fusion processes based on improved fusion specifications, and (b) to be utilized within the rules of the similarity functions, and as features in the learning algorithms, in order to improve the accuracy of the automatic fusion and validation action recommendation mechanisms.
- **Learning mechanisms for fusion and validation action recommendation**. To support automatic recommendation of validation and fusion actions, we implemented an initial learning mechanism for addressing the two major tasks we consider, as classification problems: (a) **validation** of the pair of linked POIs, deciding whether it should be accepted or rejected; (b) selection of the most **fitting action** from a set of available fusion actions. The current implementation works auxiliary to the validation and rule specification mechanism, by identifying and annotating in the fused datasets ambiguous POI links and fusion results. Our method exploits an extended set of training features that we defined and implemented specifically for the task of POI fusion and validation actions classification/recommendation.
- **Integration with the SLIPO Workbench**. FAGI v2.0 realizes two deployment modes: (a) standalone, as an individual software that accepts as input linked POI datasets and provides as output a fused POI dataset; (b) deployment within the SLIPO Workbench, where FAGI serves as an integral component of the SLIPO toolkit, and is loosely integrated by the SLIPO Workbench with the other software components into forming POI integration workflows.
- **Scalability**. In FAGI v2.0 we have implemented a simple, yet efficient in our setting, distributed execution scheme, that functions independently of the core fusion and validation functionality of FAGI. Specifically, we have implemented a component that partitions the two input POI datasets into subsets, based on subsets of the input links between POIs. Then, for each subset of the input datasets, a separate instance of FAGI is deployed. Finally, the output of all FAGI instance is merged in a final output containing all fused POI entities.

A detailed presentation of FAGI, our development roadmap, and milestone releases, are provided in the following Section 2.

2. The FAGI Framework

FAGI is currently the *only software framework* that supports a rich set of facilities for fusing geospatial Linked Data. FAGI v1.0 was initially developed in the context of the GeoKnow [GeoKnow] project, with the aim to offer a generic purpose functionality for fusing geospatial entities, considering both thematic and geospatial properties. FAGI v1.0 focused largely on facilitating manual fusion operations and provided a map-based user interface where the user was able to manually specify fusion actions per matching properties and apply them in individual pairs (or batches) of linked geospatial entities. Further, the first version of FAGI included limited functionality for fusion action recommendation and did not specialize on POI entities.

In the context of the SLIPO project, FAGI was significantly enhanced and extended to support commercial-level requirements for the efficient, accurate, scalable, and automated fusion of large POI datasets. Although a considerable amount of core fusion functionality was maintained from version 1.0, several modules of the framework were extended, specialized, enhanced, refactored, or even deprecated, in order to produce a new platform that satisfies the requirements of the SLIPO project, and specifically, of our industrial partners.

In the following, we first present our starting point, FAGI v1.0, a map-based platform that supports extended, manual fusion of geospatial Linked Data. Next, we outline FAGI's development roadmap towards its final version 3.0, planned for the end of the project. Finally, we briefly present an overview of the features already available in the current version 2.0 of the software.

2.1. FAGI v1.0

The initial version of FAGI (v1.0) allows the fusion of interlinked geospatial entities via a map-based interactive interface. It is designed to retrieve data through SPARQL endpoints, and as such it receives as input RDF graphs stored in an RDF store. This allows FAGI to operate on several RDF datasets, as long as they are served by a SPARQL endpoint. Specifically, the input of FAGI comprises two RDF datasets containing geospatial entities, and a third file that contains the links between these entities. FAGI v1.0 supports fusion of both thematic and spatial properties between pairs of linked geospatial entities, producing a single, consolidated (fused) geospatial entity for each pair of linked entities. Thus, the output of FAGI is an RDF file containing the fused geospatial entities, which essentially translates to a set of fused properties for the entity.

FAGI v1.0 [GeoKnowD323][GVK+15] implements fusion as a manual, two-step process:

1. First, the input datasets are analyzed in order to discover how geospatial properties of input entities are represented, along with their thematic properties. The interface presents a list of properties of the linked entities from the two data sources and assists the user into identifying matching properties. This is achieved by comparing the property names with respect to their textual and semantic-conceptual similarity and producing matching scores for each pair of properties.
2. Given the above property matchings are identified, a series of fusion actions are available per pair of matching properties, regarding both spatial and thematic properties of the linked entities. The

available fusion actions are enumerated in Table 1. Apart from fusion actions on the properties of individual pairs of linked geospatial entities, the user is able to perform batch fusion actions on selected subsets of entities, as well as map-based link discovery in the neighborhoods of POIs.

Action	Type	Description
Concatenation	Geometric	Create a GEOMETRYCOLLECTION Well-Known-Text from the two geometries
ShiftAtoB/BtoA	Geometric	Shift one geometry in the direction of the other' s center
Keep A/B	Geometric	Keep only the geometry of one of the datasets
Keep most points	Geometric	Keep most complicated geometry
Keep Both	Geometric	Keep both geometries separately
Keep Concatenated	Thematic	Keep a concatenation of the values corresponding to each property in one single property
Keep A/B	Thematic	Keep only the value from one of the datasets
Keep Both	Thematic	Keep both separate values
Keep largest	Thematic	Keep the largest value (string)
Keep Flattened A/B	Thematic	With properties that are part of a long RDF triple chain this action allows for creating triples that move the values of the selected property one depth level less in the chain.
Keep Flattened Both	Thematic	Perform the above action on both properties and keep both results

Table 1: FAGI v1.0 supported fusion actions

FAGI v1.0 consists of two components, corresponding to two different deployment modes: FAGI-cli and FAGI-service. The former offers a command-line interface for basic fusion functionality, while the latter provides the full-fledged, map-based, interactive user interface and functionality for geospatial entities previewing and fusion action execution. Figure 3 presents a view of the map-based component of FAGI v1.0. Through the interface, the user is able to match properties between the two datasets, select fusion actions per pair of matching properties and align (conflate) geometries directly on the map.

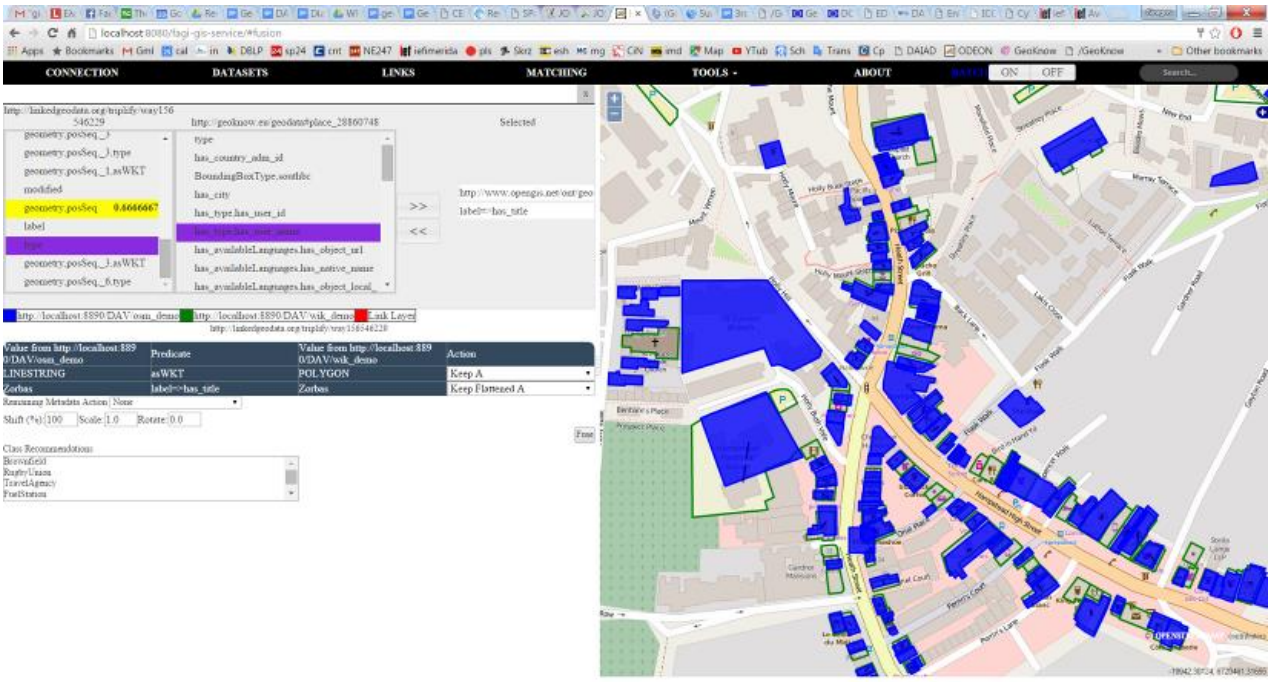


Figure 3: FAGI v1.0: property matching and fusion selection Interface

2.2. Towards FAGI v3.0

In this section, we describe the planned extensions on FAGI in the context of the SLIPO project, towards evolving it into a platform for efficiently and effectively fusing large datasets of linked POI entities. Towards this, we have revised and adapted our original development roadmap (see *D1.2 Architecture, Section 2.6.2, [SLIPOD12]*), introducing v2.0 as an interim version of the software, and increasing the version of the final anticipated software to 3.0. This update reflects the dynamic development of the software and the multitude advances we have managed to introduce already in the project's lifecycle.

Overall, the proposed extensions focus on the following development pathways:

- Adaptation and specialization of similarity functions, conditions, actions and rules on handling the fusion of POI entities;
- Incorporation of quality metrics, functions and indicators within the fusion process;
- Semi-automation of the fusion process;
- Parallelization of processing to satisfy the scalability requirement of fusing millions of linked POIs.

In the following, the planned extensions and new functionality of FAGI are presented in detail:

- **Specialization on POI data**. We will explore the available commercial datasets and schemas, to extract patterns relating to naming of POIs, frequent property types, existence of potential temporal or quality metadata, etc. In parallel, we will consider the work done in WP2 regarding the SLIPO ontology. Our main goal will be to identify peculiarities and specific characteristics of POI data, to

be used in the development of fusion action functionality. Finally, POI-specific quality indicators and quality-based rules will be identified and implemented. [Initial release: M18, Final release: M36]

- **Quality and evolution based fusion**. We will develop rules and fusion actions criteria based on quality characteristics and provenance information of the datasets, as well as based on the evolution of the linked POI entities. These will include metrics used to compare the similarity of the properties between two linked POIs and to measure and compare the quality of those metadata. Further, quality assessment and dataset comparison metrics will be integrated, not only to validate the fusion quality, but also to assist and guide the fusion process, e.g., by considering dataset quality indicators in fusion action recommendation. Finally, we will incorporate processes for including provenance and evolution of POI entities and metadata during the fusion process. The provenance will be used as one of the indicators for deciding which metadata to maintain and which to discard. On the other hand, we will implement new, evolution-specific fusion actions for handling metadata that change over time and through different, interlinked datasets. [Initial release: M18, Final release: M36]
- **Integration with the SLIPO Workbench**. Two basic deployment scenarios of FAGI will be supported: (a) (almost fully) automated deployment where human interaction is limited to the specification of a deployment configuration; (b) a more interactive version, where the user can perform manual fusion actions, train the offline learning mechanism, provide input in the active learning process and view/validate/rectify the produced fused POIs. For the latter, we will extend and adapt the existing map-based user interface, so that it supports several facilities, such as: visualization of Linked POI entities on the map; full demonstration of the properties of a selected POI in an elegant way; selecting/validating/rectifying fusion actions per properties. [Initial release: M18, Final release: M36]
- **Development of learning mechanisms for the automation of fusion action selection**. There will be three learning scenarios that we will consider: (a) offline training of fusion actions based on an initial set of fused POIs; (b) active learning of fusion actions where the system progressively trains, suggests fusion actions and re-trains based on user validation; (c) offline re-training where the system has produced a set of fused results, the user examines and validates/rejects each of them and then the system uses this feedback to "correct" its model. [Final release: M36]
- **Scalability**. FAGI v1.0 does not implement any parallel processing techniques. However, parallelization of the fusion process is straightforward, since each fusion action is performed individually in very small, autonomous fragments of the data (pairs of linked entities along with all properties attached to them). Each pair of linked entities, with all their properties, is included in only one processing node, with no impact on fusion, as each pair of entities is handled independently from the rest. Thus, we can apply the desired fusion actions separately in each node and combine the final results in a complete fused graph/RDF file. This allows us the flexibility to select among several approaches and frameworks for parallelized processing: from adopting well-established frameworks (Hadoop, Flink, SANSA) to implementing a lightweight and resource-efficient, ad hoc partitioning mechanism. Our final choice will be based on the combined targets of maximizing scalability and facilitating integration with the SLIPO Workbench. [Initial release: M18, Final release: M36]

- **Performance**. In parallel, several performance-oriented enhancements will be implemented, that will further contribute to the efficiency/scalability improvement of FAGI: (i) We will restructure the backend of FAGI, so that it gathers all basic fusion functionality, providing a common API to both the automatic, batch version and the user interactive, map-based version of the software; (ii) We will minimize spatial processing and storage dependencies, uncoupling the current version of FAGI from unnecessary frameworks and libraries, as well as removing intermediate processing steps and maximizing the in-memory processing functionality. [Initial release: M18, Final release: M36]

Finally, we have introduced the following two significant milestones for the SLIPO framework regarding the implementation of all features of our roadmap:

(a) FAGI v2.0 (M18) that supports specification of configurable rules for automated link validation and fusion action of large sets of POs, POI-specific fusion functionality, extraction of statistics and quality indicators on the input and output datasets of the fusion process, an initial learning framework for automatic link validation and POI fusion, and an initial degree of distributed execution of the fusion process;

(b) FAGI v3.0 (36) that extends the machine learning/automatic fusion action recommendation functionality, enhances the statistics/quality indicators extraction and utilization mechanisms, as well as implements a fully distributed, and thus scalable, platform for large POI datasets fusion.

2.3. Current Features and Functionality

Table 2 outlines the major functionality supported by the two successive versions of FAGI: the initial one (FAGI v1.0) that comprised our starting point in SLIPO (M0); and the intermediate version of FAGI (v2.0) that is currently available in M18. In what follows, we elaborate on the extended support of FAGI v2.0, for the most important of the features in the table.

FAGI Functionality	ver.1.0 (M0)	ver.2.0 (M18)
Fusion actions for thematic properties	6	5x2
Fusion actions for geospatial properties	5	7x2
Link validation actions	NO	YES
Fusion/validation actions based on multiple properties	NO	YES
Fusion/validation actions specialized on POI characteristics	NO	YES
Fusion rules specification	NO	YES
Link validation rules specification	NO	YES
Handling of ambiguous links and fusion actions	NO	YES
POI specific rules, conditions and functions	NO	YES
Dataset level fusion actions	PARTIALLY	YES
Batch fusion functionality	PARTIALLY	YES

Supported dataset output modes	2	4
Supported input sources	1	2
Supported input RDF formats	1	9
Supported output RDF formats	1	9
Support of POI specific vocabulary	NO	YES
Semi-automatic fusion functionality	LIMITED	YES
Semi-automatic link validation functionality	NO	YES
Extraction of Quality Indicators and Statistics	NO	YES
User Interface	MAP-BASED/ORIENTED ON MANUAL USER-INTERACTION	UI FOR FUSION RULES SPECIFICATION /ORIENTED ON AUTOMATED, BATCH FUSION
Scalability	LIMITED	IMPROVED/ CUSTOM PARTITIONING SCHEME

Table 2: Functionality supported by successive versions of FAGI

Fusion actions. FAGI currently supports a rich set of fusion actions, inherited from its previous version. This set is described in detail in Section 4.1.1.1, and includes: keeping only one of the two property values, merging the values, keeping both separately, as well as some advanced fusion functionality on the geometries of the POIs. FAGI v2.0 further enhances the fusion related facilities by additionally defining *default* fusion actions and *dataset-level* fusion actions functionality. With the former, FAGI handles all remaining matching property pairs that are not covered by the user-defined fusion rule specification. With the latter, the same issue is handled in the whole dataset level. Finally, FAGI v2.0 additionally handles the ambiguity of POI links and fusion actions. Specifically, for each initially supported fusion action, FAGI incorporates a counterpart fusion action, that executes exactly the same fusion functionality, but additionally marks the fusion output as ambiguous. This way, the fused POIs can be optionally identified and re-examined in a later, manual validation step.

Fusion rule specifications. This feature comprises a completely new functionality, towards the direction of automation and batch handling of large POI datasets. The specific module allows the definition of fusion rules via XML specifications. Fusion rules are comprised by: *expressions* of combined *condition functions* that evaluate several characteristics of the properties of two linked POIs, and *fusion actions* that are performed if the expressions are evaluated to true. The individual condition functions are combined through the "AND", "OR" and "NOT" logical operators, in order to form an expression. This way, several aspects of two linked POIs can be examined in order to decide a specific fusion action. The currently implemented conditions are enumerated in Section 4.1.1.2. Indicatively, condition functions may examine the similarity of two strings, normalize and compare two phone numbers, examine if a property has an empty value, examine the data type of a property value, etc.

In order to compose such rules, a simple XML syntax is used, where the user is expected to provide the properties to be fused, the conditions to be evaluated (and the respected properties for each condition), the

way the conditions will be combined into expressions and the respective fusion actions. A user interface is also provided that allows the graphical construction of such rules. The user is allowed to define an arbitrary set of fusion rules, considering any pair of matching properties between linked POIs, ranging from a simple condition on one property value, to arbitrarily large combinations of conditions, that form complex rules, on several property values. These rules are evaluated with sequential priority on the whole set of input POI data. This allows the realization of significantly automated fusion workflows, on massive amounts of POI data, with the user input required only at the beginning of the process (rule specification). Detailed description of the format of rule specifications and indicative examples are provided in Sections 4.1.1 and 5.3.

Link Validation functionality. In the context of SLIPO, FAGI is being extended to also serve as a quality assurance software, since such functionality is tightly interleaved (even necessary) to data fusion. Part of the quality assessment functionality is assessing the correctness of input links between POIs. Naturally, this is a task that is heavily domain and dataset dependent, and requires producing results of high accuracy. Section 3 presents in detail our work on developing similarity functions and machine learning models, specifically for POI data. Further, link validation functionality is incorporated into FAGI in a similar way to fusion action functionality: link validation rule specifications can be defined by the user using a similar XML syntax, and via a dedicated user interface.

Specialization on POI data. A core task that is performed in several stages and deployment modes within FAGI v2.0 is comparing the values of two matching properties in a pair of linked POIs and calculating a similarity value between them. Examples include several condition functions, as well as the methods we have implemented for automatic link validation and fusion action recommendation. To this end, we have examined and analyzed real world, commercial datasets, in order to develop similarity functions tailored to the characteristics of real world POI data. Section 3 provides an extended analysis on this part of our work.

Supported dataset output modes. Through the requirements elicitation process, the definition of detailed use cases and the prescription of the SLIPO pilots, we identified additional requirements on the different outputs FAGI is expected to provide, in real world deployment use cases. Specifically, FAGI v2.0 provides a set of options for handling, not only the fused POIs, but also the POIs from the initial datasets that were eventually not fused or were not linked at the first place. For example, FAGI allows the output of fused POIs on a new, separate dataset, or into one of the initial (input) POI datasets; further, it allows maintaining either within the fused dataset, or separately, the remaining, unlinked/unfused POIs. The different dataset output modes are described in detail in Section 4.1.5.

Semi-automatic fusion and link validation functionality. One of the most important challenges in the development of FAGI is to provide mechanisms for alleviating a heavily manual and user-interactive process, such as fusion. These inherent characteristics of the fusion process are enforced by the requirement of producing highly accurate results, at least on some important POI properties (name, coordinated, category, phone). To this end, FAGI v2.0 incorporates mechanisms for automatic link validation and fusion of POI properties. In Section 3, we thoroughly describe these mechanisms, which are comprised by a similarity-based and a supervised learning method for link validation, as well as a supervised learning method for fusion action recommendation. This initial development iteration of these methods provides satisfactory accuracy results, surpassing the results of baseline methods. However, working towards the improvement of accuracy is a constant, ongoing work within the development of FAGI.

3. Optimizing fusion of POIs

As a first step towards implementing the core mechanisms of fusion for POIs (*property similarity functions, training features, fusion conditions and rules and fusion actions*), we performed **an analysis of real world, commercial POI datasets**. The analysis was significantly based on the feedback of the commercial partners of the project, which provided annotations (*training labels*) on samples of the datasets, regarding fusion and validation actions. By exploring both the individual POI entities, and the linked POIs in conjunction with the provided training labels, we extracted an initial set of patterns regarding the structure and the contents of POI names. Further, we produced a set of assumptions on how the structure and contents of a POI's name affect the POI's relation with other POIs, with respect to validation and fusion.

The second step of our work consisted in exploiting the knowledge we obtained from the aforementioned process in order to **build similarity functions and rules**, and identify proper weighting schemes and thresholds for facilitating the manual construction of elaborate fusion rules. This part of our work targeted the facilitation of the *unsupervised*, rule based validation and fusion process. In this setting, the system user, having a strong domain knowledge and knowledge of their specific POI data, needs to be able to define elaborate fusion and validation rule specifications, combining condition expressions on different pairs of matching POI properties and properly adjusting similarity weights and thresholds.

As a third step, we **transformed the extracted patterns and assumptions into training features** to be used in machine learning models for automatic validation and fusion action recommendation. Towards this, we defined and implemented a series of features, representing both the characteristics of POI property values and the relation between the values of matching POI properties. These training features were assessed on a series of machine learning algorithms for classification, in order to select a robust combination of features and ML algorithms to be adopted in FAGI.

The aforementioned, iterative steps of analysis, development and experimentation allowed us to fine-tune the fusion and validation mechanisms on real world data, and achieve high accuracy in validating and fusing linked POIs. Further, it facilitated the development of learning mechanisms that allow the automatic recommendation of validation and fusion actions.

3.1. Analysis of linked POI datasets

The first steps of our work consisted in the exploration and analysis of the commercial POI datasets at hand. In this task, we did not aim at generic-purpose analysis (*e.g., for describing the datasets and providing generic statistics*), but rather at identifying interesting patterns in the data, and especially in the names of POIs, that could be exploited in the development of validation and fusion rules and learning mechanisms.

3.1.1. Problem formulation

FAGI receives as input a pair of POI datasets, A and B which contain POI entities accompanied with several properties describing characteristics of each POI (e.g., name, address, phone, website, email, coordinates), as well as a links file L, which contains a set of links connecting POIs from A with POIs from B. Considering

a POI *a* in A that is linked with a POI *b* in B, and the sets of properties {$P_{a,i}$} and {$P_{b,i}$} that describe them respectively, FAGI handles two tasks:

- **Decide validation action**. Decide whether POIs *a* and *b* actually correspond to the same real-world entity or they are wrongly interlinked. This can be formulated as a binary classification problem with output classes "accept" and "reject".
- **Decide fusion action**. Decide which is the most fitting fusion action for each pair *i* of matching properties ($P_{a,i}$, $P_{b,i}$) of the two POIs. This can be formulated as a multi-class classification problem, with the different fusion actions being considered as the classes of the problem. The major classes we have considered for experimentations and fine-tuning in FAGI are:
    - **Keep A.** Keep the property values that is corresponds to POI a.
    - **Keep B.** Keep the property values that is corresponds to POI b.
    - **Keep the largest/most complex description.** Keep the value that contains the most information. E.g., keep the longest name string, or keep the polygon geometry that is comprised by the most points.
    - **Keep both as separate properties.** Keep two distinct property values for the POI, coming from both sources.
    - **Concatenate and keep as one property.** Merge the two distinct values into one and keep them as one value. E.g., concatenate the two name strings and keep them as one string, or create a geometry collection by the two distinct geometries and keep this collection in one property.
    - **Reject (do/keep nothing).** This action might be selected when none of the above options is preferable. For example, a user is not satisfied with neither of the property values and wants to provide a completely different/new value for the property, at a later stage.

Our goal is to develop the functionality that facilitates the *accurate* execution of the above two tasks, requiring the minimum effort by the user both in the selection of validation/fusion actions and in the post-processing, manual examination/validation of the fusion results.

3.1.2. Analysis process

Our analysis starts by identifying *proper* pairs of linked POI entities. In our case, "proper" means "ambiguous": pairs of POI entities that: (a) it is not straightforward to decide whether they should be linked and (b) it is difficult to decide which value is the ideal one to keep for a respective property of the final, fused POI entity. In order to identify such pairs of POI entities, we consider the major properties that define a POI: its name and its location (*in the form of latitude-longitude coordinates*). In theory, it is highly probable that two POIs with very similar names and very close coordinates correspond to the same real-world POI entity. However, as we will see next, this is not always the case. The coordinates might be erroneous or inaccurate, due to several reasons: errors and loss in precision due to the mapping processes and hardware used, inaccurate coordinates extracted through geocoding, different conventions adopted from different vendors (e.g. the POI coordinates are defined exactly on the street vs. next to the street), etc. On the other hand, two POIs might have very similar names, and yet be different. This is commonly encountered e.g., when POIs include

in their name the name of an area or their category, or when two POIs are somehow related but are different POIs (e.g. POI within a POI).

Given the above, in order to gather a set of ambiguous pairs of POIs, we consider two POI datasets with a basic set of common attributes per POI: *coordinates, name, address (street and number), location, phone, website, email and category.*

Our initial analysis is performed on the following two POI datasets: dataset A corresponds to a TomTom POI dataset for Austria and dataset B corresponds to the Yellow Pages (Herold) POI dataset provided by WIGeoGIS, again for Austria. A short presentation of the main characteristics of the datasets is provided in the Annex (Section 8.1).

As a first step, and in order to limit the search space out of the whole POIs from the two datasets, we focus on the area of Vienna, which expectedly has a very high concentration of heterogeneous POIs. Then, POIs from datasets A and B are interlinked via a very simple criterion, that serves our purpose:

```
return pairs of POIs that are no further than 10 meters to each other
```

Note that we do not aim to find an optimal distance threshold for identifying correct links of POIs. The specific restriction threshold for the distance (10 meters) is intuitively selected so as to provide us with enough pairs of POIs for both classes of the validation task: correct and wrong links of POI entities. A larger threshold (e.g., 20-50 meters) would probably serve our purpose, providing though a much larger set of result pairs to explore. On the other hand, a smaller threshold (e.g., 5 meters) would exclude from our analysis several useful pairs of POIs.

Along with the aforementioned properties, the query we formulate also returns an indicative similarity score on the name properties of each returned POI pair. In particular, we apply the `trigram` similarity. This score is utilized as an auxiliary measure to identify and focus on ambiguous pairs of POIs. Further, in order to keep the resulting dataset of pairs of POIs in a moderate size that allows its exploration, we set extra restrictions that limit the number of results of the posed query. Specifically, we restricted the polygon of our search to focus on a smaller, central area of Vienna and limited the number of returned, intermediate results (i.e., results before linking POIs by distance). In our case (TomTom vs. WIGeoGIS-Herold POI dataset for Vienna), this provides around 10K pairs of POIs to be explored. A sample of two such resulting pairs of POIs is provided in the table below. For visibility purposes, we have separated the result records into separate sub-tables. We also note that the actual string values have been modified in order to preserve the confidentiality of the datasets.

distance	Name_A	Name_B	NAME FUSION ACTION
0.056876462	Gemeindeamt der Marktgemeinde Cardeln	Marktgemeinde Cardeln	Keep Left
0.056876462	Gemeindeamt der Marktgemeinde Hard	Energieberatung Marktgemeinde Hard	Keep Right

Stname_A	Stname_B	Stnumber_A	Stnumber_B	ADDRESS STREET FUSION ACTION
Ludfordstraße	Ludfordstrar e	38	38	Keep Left
Ludfordtraße	LudfordstrarŸe	38	38/Neuhaus	Keep Left

Phone_A	Phone_B	TELEPHONE FUSION ACTION	Email_A	Email_B	EMAIL FUSION ACTION
+(43)-(5574)-6970	05574/697-0	Keep Left	cardeln@ cardeln.at	cardeln @ cardeln.at	
+(43)-(5574)-6970	05574/697-43	Keep Right	cardeln @ cardeln.at	karim.schelde@ cardeln.at	Keep Right

Website_A	Website _B	WEBSITE FUSION ACTION	Name_score	ACCEPT OR REJECT THE PAIR OF LINKED ENTITIES
	http://www. cardeln.at		0.655172408	ACCEPT
	http://www.kodarlberg.at		0.422222257	REJECT

Table 3: Sample links of POIs annotated with fusion and validation actions

The set of POI pairs are organized in rows containing all the aforementioned metadata. Additionally, next to each pair of matching property values from the two datasets, there exists an extra field for providing a label for the desirable fusion action (e.g., "NAME FUSION ACTION"); the last field of each row is dedicated for providing a validation label on the pair of POIs ("ACCEPT" or "REJECT"). The rows corresponding to pairs of POIs are sorted in ascending distance order. Additionally, each row contains the trigram similarity of the values of the two POIs for the *name* property.

3.1.3. Analysis outcomes

After thoroughly exploring the produced dataset, we identified the properties **POI name** and **phone** as the most challenging ones, with respect to both the variability in fusion actions that fit each example pair of POIs and their importance on deciding the validation action of a POI. Before we delve in more details on these two property types, we briefly enumerate our observations on the rest properties:

- **Address name** and **number** are usually matching in most linked POIs, so the fusion action is straightforward. In the very few cases that the address name differs, usually both name values are kept in separate properties.
- **Website** and **email** also usually match exactly, however, in the case of these two properties, it is very often that one or both values are missing. Again, in these cases, the fusion action is straightforward, and when the values exist but differ, usually both values are kept in separate properties.

3.1.3.1. POI names

In the following table, we present the outcomes of our analysis considering the names of pairs of POI entities. We remind that we examine mostly ambiguous pairs of POIs, meaning that some of them actually constitute correctly linked POIs and some are wrongly paired. Each row of the table contains an observation

on a characteristic/relation of the POI names that applies on several of the observed pairs, as well as an assumption-direction on how this characteristic affects our fusion and validation tasks and how it can be handled.

Observation	Assumptions	Examples
Several names contain abbreviations that may also differ in their exact syntax	Abbreviations need to be identified at a first stage, before fully comparing the two name values. The abbreviations should be replaced by their full version and potentially be handled as special terms, separately from the core terms of the name.	"Dr" vs. "Dr." vs "Ddr"
Several names contain acronyms that may also differ in their exact syntax	Acronyms need to be identified at a first stage, before fully comparing the two name values. The acronyms should be replaced by their full version and potentially be handled as special terms, separately from the core terms of the name	"LKH" vs. "Landeskrin-derhauses"
Several names contain dashes that denote different semantics of the name	Dashed separated terms need to be specially handled when sorting the terms of a name	"Urkel Sannon-Woden Band Frecke - Ihr Hoffstalter"
Several names consist of the core POI name and several auxiliary terms that may better determine the POI	The applied similarity functions need to be able to identify and distinct between the core POI name and the auxiliary terms. To this end, the identification of special/frequent terms within the dataset, as well as terms from other POI properties (e.g., POI city/area, POI category) need to be exploited as filters. In case of actual matching of the POIs, usually the longest description is preferable to be kept in the fused POI.	"KMV" vs. "KMV Tankstelle - Daan Tankstellen GmbH & Co KG"
Several POIs refer to just an area, thus having high similarity to actual business POIs that include the area term in their name	In this case, the link between POIs should be rejected. To identify such cases, tuned similarity functions need to be used	"Böchst" vs. "Pfarramt Böchst"
The order of the terms that compose a name may vary	The terms of each name need to be alphanumerically sorted before the comparison with another name. Then, the similarity of two names need to be produced by pairwise comparison of individual terms by parsing the sorted name strings. This process should also take into account the fact that one of the names may miss a single term, which may disrupt the ordered comparison process.	"Dr.Med. Miroslav Kolte" - "Kolte Miroslav, Dr. med. univ."
Several POIs are presented either by the name of their owner, or by the actual name of the business they represent, with small or no coverage between the two namings	In this case, secondary properties (phone, email, address) need to be exploited. In the general case, both namings can be kept separately in the fused POI entity.	"Pension Bischnach" vs. "Bischnach Katrina"
Several POIs have shut down with new POIs taking their place	This is a POI timeliness issue. One way to handle it is to identify which of the two linked datasets is expected to contain the most up-to-date information, and by default use values from it	"Cafe-Restaurant Boll" vs. "Panama Restaurant"
Several POIs correspond to	In this case, the POIs need to be kept separately. Small differences in the core POI names need to be	"Dr.Med. Panya Koslov" vs.

professionals collocated to the exact building and using the same phone	identified, since most other properties are expected to contain exactly matching values.	"Koslov Gabriel, DDr."
Several POIs are located within broader POIs and share part of their names	In this "POI within POI" case, different terms of the names need to be weighted differently, in order to distinguish the POIs	"Kioskbirn-Hessepark Hessestraße" vs. "Kaprinski Boutique Sufelberg im Hessepark"
Several POI names contain just the name of the area, and the actual POI category is implied	In this case, the POI categories that are related to such naming conventions need to be identified and handled specially.	"Melission" vs. "POLICE STATION OF MELISSIA"

Table 4: POI name specificities

3.1.3.2. POI phones

In Table 5, we present the outcomes of our analysis considering the phones of pairs of POI entities. The structure of the following table, is identical to Table 4.

Observation	Assumptions	Examples
POI phone values from different sources follow different formatting conventions	Identifying prefixes/area codes need to be applied on each data source specifically. More elaborate pattern matching needs to be applied in order for such methods to generalize to several countries	"+(43)-(5522)-457249150" vs. "05522/45724915"
Phone suffixes may have different semantics in different countries	This can be handled either by pattern matching rules, or by country specific, fixed rules.	
Phone suffixes imply "POI within POI" or "POI part of POI" relations	This information can be used to decide which validation or fusion actions to be followed for the respective linked POIs.	"+(43)-(5574)-526311" vs. "05574/526311-23811"

Table 5: POI phone specificities

3.2. Similarity-based link validation

The first task that needs to be handled by FAGI is link validation. Our first approach is to solve this problem is by implementing a similarity-based method, an approach followed in several works in the literature on geospatial entities linking. Specifically, our goal has been to design a complex similarity measure that can be used to accurately identify/distinguish between correctly and wrongly linked POIs.

3.2.1. FAGISimilarity

The method we designed and implemented, called *FAGISimilarity* comprises several string processing steps, a limited set of parameters-weights, a baseline similarity function that is applied on the names of the POIs and a similarity threshold that can be identified at an initial fine-tuning stage. It has been developed by: (a) initially exploring the datasets at hand and defining a set of initial rules, and (b) through iterative applications,

assessments and enhancements based on the labelled datasets and the results that each version of FAGISimilarity achieved.

3.2.1.1. Similarity definition

Next, we present the major steps that comprise FAGISimilarity. The listing below provides a high-level pseudocode.

```
Method: FAGISimilarity

Input : nameA, nameB, threshold

Output : TRUE||FALSE

1 (nameA, nameB) = IDENTIFYANDEXPANDABBREVIATIONS(nameA, nameB)

2 (nameA, nameB) = REMOVECHARACTERSANDLOWERCASE(nameA, nameB)

3 (nameA, nameB) = SORTTERMSWITHINSTRING(nameA, nameB)

4 (nameA, nameB, nameSpecialA, nameSpecialB) = EXTRACTSPECIALTERMS(nameA, nameB)

5 (nameBaseA, nameBaseB, nameMisA, nameMisB) = COMPAREANDSPLITNAMES(nameA, nameB)

6 matchingScore = WEIGHTEDSIMILARITY(nameBaseA, nameBaseB, nameMisA,
                                     nameMisB, nameSpecialA, nameSpecialB)
7 return (matchingScore ≥ threshold)
```

The first steps of the similarity (*line #1, line #2*) consist in cleaning and normalizing the two strings (names) to be compared, which is a very common step in relevant tasks. Further, the terms within the strings are sorted alphabetically, to increase the probabilities of two matching terms between the strings to be found in the same position in the name (*line #3*). Then, we extract frequent terms from the whole available dataset and handle them separately, when we identify them within the compared strings *(line #4*). Specifically, for each "base" name string that is compared, we define an auxiliary string that is filled with the identified frequent terms in the base string.

Going back to alphabetical sorting of terms in the base string, this step is justified by the fact that it is quite often the case that specific terms of a name are shuffled within it. For example, the term "Dr." might be found in several positions within the name of a POI designating the business (medical practice) of the specific doctor. On the other hand, the shuffling of terms can be handled by specific meta-similarities, like the Monge-Elkan similarity, which identifies the pairs of terms between the two names with the highest similarity. However, such types of similarities need to compare all the terms of the first string with all the terms from the second one, something which becomes inefficient as the number of terms increases, since it has quadratic complexity. To avoid this, we define a custom parsing scheme, that considers two cursors, initially placed at the beginning of the two sorted base strings to be compared. Each cursor is able to move by one term right at each step. At each step, a similarity comparison is performed between the two terms where the cursors are currently placed. If the terms are found similar enough (considering a loose threshold combined with Jaro-Winkler similarity), the terms are maintained in the base string and the two cursors are

simultaneously moved to the next position. If the terms are not found similar enough, only the cursor that corresponds to the term that begins with the alphabetically lower character is moved one position to the right, while the corresponding term is moved to an auxiliary string called "mismatch" string (*line #5*).

Through this process we ensure that: (a) the base strings for the two POI names end up containing only (even loosely) matching terms, and (b) this process is performed linearly to the number of terms of the strings, instead of quadratically. Eventually, three types of strings (or lists of terms) are considered for each pair of compared POI names: *base string, mismatch string and special (frequent) term list*. For each of these strings, we extract a similarity score. To do so, any baseline similarity function can be used, as shown next. The three individual scores are weighted and combined into the final similarity score, normalized in the interval [0-1] (*line #6*). The final score is compared with a threshold and if the score is higher, then the link between the pair of POIs is accepted; otherwise, it is rejected.

FAGISimilarity can be characterized as a "meta-similarity" function, in the sense that it prescribes a series of operations to be performed, utilizing more basic similarity measures in several steps. In our method, we support the utilization of the following basic similarity measures:

- Levenshtein
- 2-Gram
- Longest Common Subsequence
- Jaro
- Jaro-Winkler

3.2.1.2. Accuracy evaluation

In this section, we present an initial set of experiments that assess the accuracy of the developed similarity function in validating (accepting/rejecting) links of POIs.

The evaluation dataset consists of 192 labelled POI pairs (links labelled as "accept" (correct) or "reject" (wrong)), from the dataset described in Section 3.1.2 (based on the datasets described in Section 8.1). The dataset is balanced (96 correct vs. 96 wrong links), so that the following evaluation measures can be more intuitively interpreted.

In our experiments, we used four measures to assess the effectiveness of the evaluated similarity functions:

- Accuracy: measures the percentage of links that where correctly classified, either as accept or as reject.
- Precision: measures the percentage of accepted links that are correctly accepted.
- Recall: measures the percentage of correct links that are correctly accepted.
- F-measure: harmonic mean of Precision and Recall.

As baseline similarity functions, we used the basic similarity functions presented in Section 3.2.1.1: *Levenshtein, 2-Gram, Longest Common Subsequence, Jaro, Jaro-Winkler*. For each similarity function, we need to define a threshold that distinguishes between accepted and rejected links. To do so, we follow the methodology of [SFM17], which comprises an extended study on the effectiveness of several similarity measures for toponym matching (interlinking). Specifically, for each evaluated function, we select the

threshold that achieves the optimal accuracy value, and consequently, we report the accuracy, precision, recall, and f-measure values corresponding to this threshold.

Our evaluation considers three modes of applying the compared similarity functions:

- **Initial POI names (baseline)**. The baseline similarity functions are applied in the initial POI names, without performing any pre-processing on them.
- **Cleared and sorted POI names (baseline)**. In this mode, before applying the baseline similarity functions, we clear the name strings from punctuation and special characters and we sort the terms within each name string alphanumerically.
- **FAGISimilarity**. This mode represents the full application of our proposed method, utilizing several of the baseline similarity functions.

The evaluation results are presented in Table 6. Note that we select the best representative of each method by considering the configuration of the method that achieved the highest *accuracy*. In case of ties in accuracy values (i.e., the same method presents several configurations with the same, highest accuracy), we also consider the highest *precision* values. Intuitively, these two measures are also the two most important for our use cases in SLIPO. Accuracy measures the overall effectiveness of the system, considering both missed links and wrongly accepted links. High precision ensures that the number of POI pairs that are wrongly accepted is minimized, thus directly affecting the quality of the produced results of the integration process.

Mode	Similarity function	Accuracy	Precision	Recall	F-measure
Initial(baseline)	Levenshtein	0.625	0.773	0.354	0.486
Initial(baseline)	2-Gram	0.641	0.69	0.51	0.587
Initial(baseline)	Longest Common Subsequence	0.651	0.641	0.688	0.664
Initial(baseline)	Jaro	0.651	0.684	0.563	0.618
Initial(baseline)	Jaro-Winkler	0.667	0.64	**0.76**	**0.695**
Sorted(baseline)	Levenshtein	0.667	0.75	0.5	0.6
Sorted(baseline)	2-Gram	0.661	0.701	0.563	0.624
Sorted(baseline)	Longest Common Subsequence	0.646	0.671	0.573	0.618
Sorted(baseline)	Jaro	0.635	0.861	0.323	0.47
Sorted(baseline)	Jaro-Winkler	0.63	0.692	0.469	0.559
FAGISimilarity	Levenshtein	**0.693**	**0.878**	0.448	0.593
FAGISimilarity	2-Gram	**0.693**	**0.878**	0.448	0.593
FAGISimilarity	Longest Common Subsequence	0.677	0.815	0.458	0.586
FAGISimilarity	Jaro	0.688	0.7	0.656	0.677
FAGISimilarity	Jaro-Winkler	0.682	0.769	0.521	0.621

Table 6: Effectiveness of similarity-based link validation based on POI names

The above table clearly demonstrates the effectiveness of applying our method in validating links between POIs. FAGISimilarity improves the effectiveness (accuracy) of **all** underlying basic similarity functions. Also, applying FAGISimilarity, with either Levenshtein or 2-Gram similarities, achieves considerably higher classification accuracy compared to the best achieving function on just sorted name strings (69.3% vs. 66.7%). Further, for the specific value of accuracy, the achieved precision is exceptionally high, reaching 87.8%, which means that the links that are accepted are mostly correct. We should note here, that the specific evaluated dataset is comprised of 50% correct links and 50% wrong links. Thus, accuracy results reported in Table 6, with accuracy=66.7%, precision=64% and recall=76%, denote a trivial behavior of the respective methods, since these methods essentially classify most instances as correct.

3.2.2. Supervised link validation and fusion

The similarity-based method presented in the previous section, however accurate and lightweight may be, suffers from lack of *generalization*, as well as from the need to *fine-tune it* by trial and error, whenever the characteristics of the linked datasets change. Further, it covers only link validation, i.e., only one of the two major tasks handled by FAGI. In order to obtain a broader framework that can cover both validation and fusion action selection, we implemented a supervised method, called *FAGILearning*. FAGILearning implements the major steps of a traditional process for classification, exploiting domain-specific training features specifically designed for handling the validation and fusion of linked POI entities.

The first step of our process is constructing an initial training dataset, which consists of labelled instances (pairs of linked POIs). This process is already described in Section 3.1.2. The second step consists in representing each pair of linked POIs in a multidimensional feature space; essentially, represent each pair as a feature vector (feature extraction). The third step regards evaluating a set of different algorithms for classification.

3.2.2.1. Training features

We define the features that are described in Table 7. In this stage of our work, we mainly focus on extracting features regarding the names of the considered, linked POIs. This decision was made due to the fact that quite often, the POI name is the *only* or the most *trustworthy* property of a POI. As such, creating a first version of our model that could be based exclusively on POI names would be beneficial in several POI integration scenarios. Consequently, almost all features presented next revolve around POI names. We consider the following feature categories:

- **Structural/semantic features**. These features try to capture structural and semantic attributes of the POI names or relations between them, e.g., if the terms of one name are contained in the other name, if a name contains frequent terms.
- **Similarity features**. These features are comprised by all the similarity measures we examined in Section 3.2.1.2, including our custom FAGISimilarity.
- **Non-name based features**. These features examine phone and address similarity between the linked POIs.

Feature Name	Feature type	Symmetric feature	Short description	Total slots in feature vector
Structural/semantic features				
A contains B	Binary	B contains A	The string of A contains all the words of the string of B	2
A contains B - Position	Nominal – 3 Boolean features	B contains A - Position	The average position in string A where the terms of B are contained (start-middle-end)	6
A contains (frequent) special term	Binary	B contains (frequent) special term	A contains a special term that is frequently encountered in the datasets	2
A contains (frequent) special term - Position	Nominal – 3 Boolean features	B contains (frequent) special term - Position	The approximate position in string A where the special term is contained (start-middle-end)	6
A contains a specific (frequent) special term	Array of Boolean values, covering all considered special terms	B contains a specific (frequent) special term	A contains a term "T": frequently encountered in our datasets.	2x(number of considered special terms)
A contains abbreviation	Binary	B contains abbreviation	-	2
A contains term in parenthesis	Binary	B contains term in parenthesis	-	2
A contains dash connected words	Binary	B contains dash connected words	The property contains words that are connected with dashes.	2
Number of words in A	Integer	Number of words in B	-	2
Plain (initial) similarity features				
Initial Levenshtein	Real – normalized to [0,1]		Normalized similarity score between the two strings according to the respective measure	1

Initial N-grams	Real – normalized to [0,1]	-	Normalized similarity score between the two strings according to the respective measure	1
Initial Cosine	Real – normalized to [0,1]	-	Normalized similarity score between the two strings according to the respective measure	1
Initial Longest Common Substring	Real – normalized to [0,1]	-	Normalized similarity score between the two strings according to the respective measure	1
Initial Jaro	Real – normalized to [0,1]	-	Normalized similarity score between the two strings according to the respective measure	1
Initial Jaro-winkler	Real – normalized to [0,1]	-	Normalized similarity score between the two strings according to the respective measure	1
Similarity on sorted strings features				
Sorted Levenshtein	Real – normalized to [0,1]	-	Normalized similarity score between the two SORTED strings according to the respective measure	1
Sorted N-grams	Real – normalized to [0,1]	-	Normalized similarity score between the two SORTED strings according to the respective measure	1
Sorted Cosine	Real – normalized to [0,1]	-	Normalized similarity score between the two SORTED strings according to the respective measure	1
Sorted Longest Common Substring	Real – normalized to [0,1]	-	Normalized similarity score between the two SORTED strings according to the respective measure	1
Sorted Jaro	Real – normalized to [0,1]	-	Normalized similarity score between the two SORTED strings according to the respective measure	1
Sorted Jaro-winkler	Real – normalized to [0,1]	-	Normalized similarity score between the two SORTED strings according to the respective measure	1
FAGISimilarity features				

FAGI Levenshtein	Real – normalized to [0,1]	-	Normalized similarity score between the two SORTED strings according to the FAGI-extended respective measure	1
FAGI N-grams	Real – normalized to [0,1]	-	Normalized similarity score between the two SORTED strings according to the FAGI-extended respective measure	1
FAGI Cosine	Real – normalized to [0,1]	-	Normalized similarity score between the two SORTED strings according to the FAGI-extended respective measure	1
FAGI Longest Common Substring	Real – normalized to [0,1]	-	Normalized similarity score between the two SORTED strings according to the FAGI-extended respective measure	1
FAGI Jaro	Real – normalized to [0,1]	-	Normalized similarity score between the two SORTED strings according to the FAGI-extended respective measure	1
FAGI Jaro-winkler	Real – normalized to [0,1]	-	Normalized similarity score between the two SORTED strings according to the FAGI-extended respective measure	1
Non-name based features				
Same Phone	Binary	-	Examines whether phone numbers are the same after normalization (removal of suffixes/exit codes)	1
Same Address Number	Binary	-	Examines whether the address numbers are the same	1
Similar Address Street	Binary	-	Examines whether the address street are similar	1

Table 7: Training features

3.2.2.2. Classification algorithms

We experimented with several classification algorithms, in both settings (link validation and fusion action recommendation). Specifically, we assessed the following algorithms:

- K-Nearest Neighbors
- Support Vector Machines (linear and RBF)
- Decision Trees
- Random Forests
- AdaBoost
- Gradient Boosting

- Extra Trees
- Neural Networks

In our experiments, we utilized the scikitlearn framework [SCIK] and executed these algorithms using their default parameterization and by applying a limited, manual tuning effort.

3.2.2.3. Evaluation of validation action recommendation

In this section, we present an initial set of experiments that assesses the accuracy of the developed learning models in validating (accepting/rejecting) links of POIs. The evaluation dataset and the evaluation measures are identical to the ones in Section 3.2.1.2. We remind that the dataset is balanced, consisting of 96 correct vs. 96 wrong links.

The accuracy, precision, recall and f-measure values achieved by the learning models that were evaluated are presented in Table 8. We note that the results are produced after randomizing the order of the instances and using a five-fold evaluation process (*four folds used for training and one fold for testing, averaging results on all five different fold configurations*).

Mode	Accuracy	Precision	Recall	F-measure
Nearest Neighbor	0.71	0.77	0.68	0.71
Linear SVM	0.76	**0.91**	0.63	0.74
RBF SVM	0.75	0.90	0.61	0.72
Decision Tree	0.77	0.81	0.76	0.78
Random Forest	0.76	0.88	0.64	0.74
Extra Trees	**0.81**	0.87	0.76	**0.81**
Neural Net	0.76	0.83	0.70	0.75
AdaBoost	0.75	0.83	0.69	0.75
Naive Bayes	0.72	**0.91**	0.56	0.68
Gradient Boosting	0.80	0.85	**0.78**	**0.81**

Table 8: Effectiveness of supervised link validation

The first observation is that the reported results are superior to the respective results of the similarity-based methods (Table 6). This is expected, since the above algorithms are able to construct more elaborate classification models, exploiting the features we defined (Section 3.2.2.1). Further, we observe that some of the models achieve rather high accuracy and precision results. Specifically, Extra Trees (extremely randomized decision trees) and Gradient Boosting achieve the higher accuracy values, 81% and 80% respectively. They also achieve the highest f-measure value with 81%. Linear SVM achieves the highest precision, with adequately high values for the rest of the measure. Finally, Gradient boosting and Extra Trees again achieve the two highest values for recall, 78% and 76% respectively.

The above results indicate that learning a proper model on POI link validation, which would incorporate even more elaborate training features and tuning/feature selection processes, can potentially yield *near-ideal* link validation accuracy results.

3.2.2.4. Evaluation of fusion action recommendation

In this section, we present an initial set of experiments that assesses the accuracy of the developed learning models in recommending fusion actions for POIs.

The evaluation dataset consists of 143 labelled POI pairs (links) from the dataset described in Section 3.1.2., i.e., links between TomTom dataset and Herold dataset for Vienna, Austria, which have been verified as *correct* links by a geomarketing expert. In the specific dataset, four distinct fusion properties apply. Table 9 presents the distribution of these fusion actions in the dataset.

Keep Left	Keep Right	Keep longest/more complete value	Keep both as separate attributes/values
15%	36%	42%	7%

Table 9: Distribution of name fusion action labels in the experimental dataset

In the specific experiment, it is only meaningful to report on the accuracy of the trained model in predicting the correct fusion action on the name property of the linked POIs, since a multi-class classification problem is handled. The accuracy percentages achieved by the models that support multi-class classification in the framework we used are presented in Table 10. We note that the results are produced after randomizing the order of the instances and using a five-fold evaluation process (*four folds used for training and one fold for testing, averaging results on all five different fold configurations*).

Algorithm	Accuracy
Linear SVM	0.52
Decision Tree	0.59
Random Forest	0.55
Neural Net	0.50
Deep Neural Net	0.58

Table 10: Fusion action recommendation accuracy

We observe that the best accuracy (close to 60%) is achieved by a Decision Tree, followed closely by a Deep Neural Network. Given that the majority class (fusion action) is applied to 42% of the instances, the reported results indicate that all assessed algorithms demonstrate non-trivial behavior, i.e., improve the naive scenario of just using the majority class as recommendation for all test instances. Although the reported results are far from ideal, they are encouraging considering that (i) hardly any algorithm tuning was performed, and (ii) no explicit feature selection was performed. In our ongoing work, we will extract and assess more elaborate training features that will potentially improve the accuracy of the models.

4. FAGI v2.0

In this section, we present in detail the FAGI v2.0 software. First, we thoroughly present the major fusion and quality assessment facilities that are currently supported by FAGI. Next, we present FAGI's architecture, including its input and output, its main modules, as well our customized partitioning scheme that allows FAGI to process large amount of POI data in parallel. Finally, we provide information regarding the utilized software libraries, and software documentation.

4.1. Major features and functionality

The major functionality of the current version of FAGI is summarized in the form of five (5) high-level feature categories, which are further elaborated in the sub-sections that follow[1].

- **Fusion** of the properties of linked POIs through the specification of fusion rules (Section 4.1.1).
- **Validation of input links** between POIs through the specification of validation rules (Section 4.1.2).
- **Recommendation** of link validation and fusion actions through learning mechanisms (Section 4.1.3).
- Extraction of **quality indicators/statistics** on the input and output datasets (Section 4.1.4).
- **Configurable** output of fusion results (Section 4.1.5).

4.1.1. POI fusion

4.1.1.1. Fusion actions

Fusion of POI properties is the core task of FAGI. Fusion actions are defined regarding both thematic and geospatial properties of linked POIs. A certain fusion action is atomically applied on a pair of matching properties between two linked POIs. Consequently, different fusion actions are allowed to be defined for different types of POI properties. FAGI v2.0 supports the following fusion actions. Note that, at the end of each feature, we denote whether it regards thematic or spatial properties of POIs.

- **keep-left**. Keeps the property value of the POI from the left input dataset, in the fused POI description. *(Thematic+ Geospatial).*
- **keep-right**. Keeps the property value of the POI from the right input dataset, in the fused POI description. *(Thematic+ Geospatial).*
- **keep-both**. Keeps both property values of the linked POIs, as separate properties in the fused POI description. *(Thematic+ Geospatial).*

[1] In the remainder of the document, the terms '**first dataset/left dataset/dataset A**' and 'second dataset/**right dataset/dataset B**' are used interchangeably to discriminate between the two input POI datasets of FAGI. Similar terminology is used for the POIs and their property values.

- **concatenate**. Keeps both property values of the linked POIs, as a concatenated literal in the same, single property of the fused POI description. *(Thematic).*
- **concatenate-geometries**. Creates and keeps in the fused POI description a *GEOMETRYCOLLECTION* geometry from the two individual geometries of the input linked POIs. *(Geospatial).*
- **keep-longest**. Keeps the longest literal (property value) of the linked POIs in the fused POI description. *(Thematic).*
- **keep-more-points**. Keeps, from the initial geometries of the linked POIs, the one that is composed by the most points. *(Geospatial).*
- **keep-more-points-and-shift**. Keeps the geometry with the most points and shifts its centroid to the centroid of the other geometry. *(Geospatial).*
- **shift-left-geometry**. Shifts the geometry of the left source entity to the centroid of the right. *(Geospatial).*
- **shift-right-geometry**. Shifts the geometry of the right source entity to the centroid of the left. *(Geospatial).*
- **keep-most-recent**. Keeps the value from the dataset that has been denoted as containing the most recent/up-to-date data. *(Thematic+ Geospatial).*

The result of a fusion action on a pair of matching properties is the replacement of these properties and their values, by the property(-ies) value(-s) that are prescribed by the fusion action.

Most of the above fusion actions were inherited from the previous version of FAGI, since they comprise the core fusion functionality that needs to be performed on geospatial datasets. With the aim to incorporate quality assessment processes within fusion, FAGI v2.0 further extends this set of fusion actions in order to take into account the potential **ambiguity** in performed fusion actions. Specifically, for each of the above fusion actions a "counterpart" fusion action is defined, that denotes that the specific fusion outcome is ambiguous and needs to be examined by an expert at a later stage of the integration. The output of the application of an ambiguous counterpart of a fusion action is: (a) the exact outcome of the baseline fusion action and (b) an additional RDF triple that denotes that the specific result is ambiguous. For example, if the ambiguity refers to the address property of a POI, the resulting RDF triple would look like this:

```
<http://slipo.eu/def#POI_URI> <http://slipo.eu/def#hasAmbiguous> <http://slipo.eu/def#address>
```

The aforementioned fusion actions are utilized within fusion rule specifications (see Section 4.1.1.3), which examine certain properties of the linked POIs and decide whether to apply the fusion action or not. Thus, it is possible that a certain fusion rule specification does not cover all possible use cases, and consequently, no fusion action from the specification can be applied. To handle such cases, *default* fusion actions and *dataset-level* fusion actions are defined. For each pair of matching properties, FAGI first evaluates all available fusion rules from the specification. If none of the rules applies for a specific pair of properties, then the default fusion action for the specific pair of matching properties is applied. If no fusion specification exists for the pair of matching properties, then the dataset-level fusion is applied. While the supported default fusion actions are drawn from the list above, the supported dataset-level fusion actions are limited to: *keep-left*; *keep-right*; *keep-both*.

4.1.1.2. Condition functions

A *condition* is a function applied on the value of a property, or on the values of a pair of matching properties of two linked POIs. It evaluates to `True` or `False` and may regard several aspects of the property's values. The currently implemented conditions are enumerated below. Note that we use the term "literal" for denoting string values of thematic properties, and the term geometry for the values of geospatial properties.

- **isSameSimpleNormalize:** Checks if the two given literals are same. It normalizes the two literals with some basic steps and uses the provided similarity (default JaroWinkler) to return True if the result is above the provided threshold. The threshold must be in the interval [0,1].
- **isSameCustomNormalize:** Checks if the two given literals are same. It normalizes the two literals with some extra steps in addition to the simple normalization, using FAGISimilarity (see Section 3.2.1) and returns True if the result is above the provided threshold. The threshold must be in the interval [0,1].
- **isLiteralAbbreviation.** Checks if the given literal is or contains an abbreviation of some form.
- **LiteralContains**. Checks whether a literal contains a specific string (*which might also be the second literal*).
- **isLiteralLonger**. Examines if the first literal is longer than the second one.
- **literalHasLanguageAnnotation**. Examines if the literal includes an annotation on the language used.
- **isLiteralsSameLanguage**. Compares the language of two literals, in case both include language annotations.
- **isLiteralNumeric**. Examines if the literal is a number.
- **isGeometryMoreComplex:** Checks if the first geometry has more points than the second.
- **isPointGeometry**. Checks whether the geometry is a point geometry.
- **geometriesHaveSameCentroid**. Examines whether two geometries have centroids close enough as to be considered the same. It requires a user provided distance threshold for the centroids.
- **isGeometryContained**. Examines whether the first geometry contains the second geometry.
- **geometriesOverlap**. Examines whether the two geometries overlap.
- **geometriesCloserThan**. Examines whether the distance of the two geometries is smaller than a user provided threshold.
- **geometriesHaveSameArea**. Examines whether two geometries have areas close enough as to be considered the same.
- **isPhoneNumberParsable:** Checks if the given phone number consists of only numbers or contains special character and/or exit code.
- **isSamePhoneNumber:** Checks if the given phone numbers are the same having performed some simple normalization steps (*removal of non-numeric characters*).

- **isSamePhoneNumberUsingExitCode:** Checks if the given phone numbers are the same, after having performed some simple normalization steps (*removal of non-numeric characters, normalization of exit code notation*).
- **isSamePhoneNumberCustomNormalize:** Checks if the given phone numbers are the same. It first applies the normalization steps of the two condition functions above. If the equality comparison fails, some custom steps for normalization are executed and the function rechecks for equality (*ignoring country code, area code and trailing digits*).
- **phoneHasMoreDigits**. Examines if the first phone has more digits than the second one.
- **exists:** Checks if the given property exists in the model of the entity.
    - **notExists:** The reverse function of exists. Returns true if the selected property is not found in the model.
- **isDateKnownFormat:** Checks if the given date string complies to a known data format. The known formats are defined at FAGI's specification constants: [FAGISPEC].
- **isValidDate:** Evaluates the given date against the target format.
- **isDateSame**: Compares two given date values.

4.1.1.3. Fusion specifications

Beyond specializing fusion actions per property type, the user might need to fuse properties of POIs conditionally, i.e., to proceed to fusion only when certain conditions are satisfied. To this end, FAGI v2.0 realizes a fusion **rule specification scheme**, based on the following concepts:

- **Pair of matching properties**. The pair of properties upon which the fusion action will be performed.
- **Condition function.** A condition is a function applied on the value of a property or on the values of a pair of matching properties of two linked POIs. It evaluates to `True` or `False` and may regard several aspects of the property's values.
- **Condition expression**. A condition expression is constructed by combining several conditions with the logical operators `AND`, `OR` and `NOT`. A condition expression also evaluates to `True` or `False` and its evaluation value decides whether a fusion action will be performed on a pair of matching properties or not.
- **External properties**. These properties are not fused, but are rather utilized within condition functions to decide on the fusion action of the properties to be fused.
- **Fusion action rules**. The fusion rule specification scheme allows the definition of arbitrary pairs of the form `(condition expression, fusion action)`, for the same pair of matching properties. These pairs, denoted as `fusion action rules`, are evaluated sequentially; thus, their priority is defined by their ordering in the fusion rule specification file.
- **Fusion action**. The fusion action is the action that will be applied on the values of the matching properties, and it is always part of an `action rule.`

- **Default fusion action.** A default fusion action is applied to the properties to be fused, in the case where none of the fusion action rules is eventually applied (i.e. all respective condition expressions evaluate to `False`).
- **Fusion rule**. Having defined the above concepts, a fusion rule is comprised by the following elements:
  a. A pair of matching properties to be fused.
  b. A set of external properties to be used in the condition functions.
  c. An ordering of fusion action rules, with each action rule regarding the same pair of properties from (a.) and being composed by a condition expression and a fusion action.
  d. A default fusion action, in case none of the action rules is eventually applied.

The total set of fusion rules, along with the dataset-level fusion action, that are defined for a specific pair of input POI datasets comprises the **fusion rule specification** for the respective input.

In Figure 4, we present a sample XML snippet corresponding to the configuration of fusion rules for a pair of matching properties.

```
<rule>
   <propertyA>http://slipo.eu/def#name http://slipo.eu/def#nameValue</propertyA>
   <propertyB>http://slipo.eu/def#name http://slipo.eu/def#nameValue</propertyB>
   <externalProperty id="a1">http://slipo.eu/def#phone
   http://slipo.eu/def#contactValue</externalProperty>
   <externalProperty id="b1">http://slipo.eu/def#phone
   http://slipo.eu/def#contactValue</externalProperty>
   <actionRuleSet>
       <actionRule>
          <condition>
              <expression>
                 <and>
                     <function>isSamePhoneNumberCustomNormalize(a1,b1)</function>
                     <function>isSameCustomNormalize(a,b,0.6)</function>
                 </and>
              </expression>
          </condition>
          <action>keep-longest</action>
       </actionRule>
       <actionRule>
          <condition>
              <expression>
                 <and>
                     <function>isSamePhoneNumberCustomNormalize(a1,b1)</function>
                     <expression>
```

```
                        <not>
                            <function>isSameCustomNormalize(a,b,0.6)</function>
                        </not>
                    </expression>
                </and>
            </expression>
        </condition>
        <action>keep-both</action>
    </actionRule>
</actionRuleSet>
<defaultAction>keep-right</defaultAction>
</rule>
```

Figure 4: Sample Fusion Rule Specification

The rule presented in Figure 4 handles the fusion of the *name* property values between linked POIs. It consists of two action rules that are evaluated in the order they are defined. The first action rule examines whether the phone of the two POIs match, and whether the names of the two POIs have a similarity larger than a specific threshold (0.6). If both conditions apply, the two name values are fused by keeping the larger string. That is, the names are considered similar enough to identify them as the same name, but potentially written in a slightly different way. The second action rule also checks whether the phones of the two POIs match, but now checks whether the name similarity is lower than the threshold. In this case, both name values are kept separate attributes of the fused POI, since they are probably different names for the same POI. Finally, in case none of the two action rules applies (i.e., their conditions evaluate to `False`), the default fusion action prescribes to keep the name value from the second (right) dataset.

The mechanism we introduced for constructing fusion rule specifications significantly facilitates the *automation* of the fusion process, since it allows the user to define both **generic rules**, that cover a wide range of trivial fusion cases (*e.g., when one of the properties has an empty value*), as well as an arbitrary number of very **specific and detailed rules**, that capture and handle more complex scenarios (*e.g., when the names of two POIs are not similar enough, but their phones are the same*). As such, a large part of the (*previously manually*) fusion process can be automated. Of course, this does not guarantee an optimal accuracy on the validated links, or on the selected fusion actions. However, it allows the data integrator to process the linked input datasets in batches and define rules of diverse granularity that will potentially relieve her from manually examining a large percentage of the fused POI data, during a later stage of quality assurance/manual validation.

4.1.2. POI link Validation

4.1.2.1. Validation actions

An important aspect of quality assurance lies in validating the fusion input and deciding whether the linked entities should be either fused, further examined, or rejected as erroneous. To this end, in FAGI v2.0 we

define a set of validation actions, as well as a validation specification scheme, for constructing validation rules and evaluating them on the input POI data. The supported **link validation actions** are the following:

- **accept-link**. The link between two input POIs is accepted and the fusion process is executed as prescribed by the user.
- **reject-link**. The link between two input POIs is rejected. No fusion processes are executed for the specific pair of POIs.
- **accept-mark-ambiguous**. The link between two input POIs is accepted and the fusion process is executed as prescribed by the user. Further, the produced (*fused*) POI is marked as ambiguous, via an additional RDF triple, for later examination. Also, the initial POI descriptions (*from which the fused POI was produced*) are kept in an auxiliary output dataset, in case they are utilized in a later examination/validation stage.
- **reject-mark-ambiguous**. The link between two input POIs is rejected. No fusion processes are executed for the specific pair of POIs. Further, the individual POIs are marked as ambiguous, via additional RDF triples, for later examination. Also, the initial POI descriptions (*whose link was rejected*) are kept in an auxiliary output dataset, in case they are utilized in the later examination of the POI.

4.1.2.2. Condition functions

A *condition* is a function applied on the value of a property, or on the values of a pair of matching properties of two linked POIs. It evaluates to `True` or `False` and may regard several aspects of the property's values. The condition functions that are utilized within validation action specifications are exactly the same with the ones for fusion action specifications, and are enumerated in Section 4.1.1.2.

4.1.2.3. Validation specifications

Validation rule specifications follow the same rationale with fusion rule specifications. Specifically, validation rule specifications consider the following concepts:

- **Validation action**. One of the four available link validation actions.
- **Condition function.** A condition is a function applied on the value of a property or on the values of a pair of matching properties of two linked POIs. It evaluates to `True` or `False` and may regard several aspects of the property's values.
- **Condition expression**. A condition expression is constructed by combining several conditions with the logical operators `AND`, `OR` and `NOT`. A condition expression also evaluates to `True` or `False` and its evaluation value decides which validation action will be performed on a pair linked POIs.
- **External properties**. External properties are the only kind of properties that are valid within validation rule specifications, since no properties are fused. The functionality of external properties is identical to the functionality of external properties for fusion rule specifications, i.e. they are utilized within condition functions to decide which validation action to perform.
- **Validation action rules**. The validation rule specification scheme allows the definition of arbitrary pairs of the form `(condition expression, validation action)`, for the same pair of linked POIs.

These pairs, denoted as `validation action rules`, are evaluated sequentially; thus, their priority is defined by their ordering in the validation rule specification file.

- **Default validation action.** It has identical functionality to the default fusion action, i.e. it is applied to the pair of linked POIs, in the case where none of the validation action rules is eventually applied (i.e. all respective condition expressions evaluate to `False`).
- **Validation rule**. Having defined the above concepts, a validation rule is comprised by the following elements:
  a. A set of external properties to be used in the condition expressions.
  b. An ordering of validation action rules, with each one being composed by a condition expression and a validation action.
  c. A default validation action, in case none of the action rules is eventually applied.

The total set of validation rules that are defined for a specific pair of input POI datasets comprises the **validation rule specification** for the respective input.

In Figure 5, we present a sample XML snippet corresponding to the configuration of validation rules for a pair of POIs.

```
<validationRule>
    <defaultAction>accept</defaultAction>
        <actionRuleSet>
            <actionRule>
               <action>reject</action>
                <condition>
                   <expression>
                      <not>
                          <function> isSameCustomNormalize(a0, b0, 0.5)</function>
                      </not>
                   </expression>
                </condition>
            </actionRule>
            <actionRule>
                <action>reject-mark-ambiguous</action>
                <condition>
                   <expression>
                      <not>
                         <function> isSameCustomNormalize(a0, b0, 0.7)</function>
                      </not>
                   </expression>
                </condition>
            </actionRule>
            <actionRule>
                <action>accept-mark-ambiguous</action>
                <condition>
                   <expression>
```

```
                    <not>
                        <function> isSameCustomNormalize(a0, b0, 0.9)</function>
                    </not>
                </expression>
            </condition>
        </actionRule>
    </actionRuleSet>
    <externalProperty id="a0">http://slipo.eu/def#name
    http://slipo.eu/def#nameValue</externalProperty>
    <externalProperty id="b0">http://slipo.eu/def#name
    http://slipo.eu/def#nameValue</externalProperty>
</validationRule>
```

Figure 5: Sample Link Validation Rule Specification

In the example presented in Figure 5, the validation rule utilizes the *name* property values between linked POIs. It consists of two action rules that are evaluated in the order they are defined. The first validation action rule examines whether the similarity of the names of the two POIs is lower than the given threshold (0.5) by using the 'not' operator in the condition expression. In that case, the names are considered dissimilar enough to reject the link (*the condition will evaluate to* `True`). If the first condition does not evaluate to `True`, the second validation action rule will be checked. If the application reaches the second condition, we know that the similarity of the POI names is over 0.5, so we want to define another condition. Thus, we construct a condition that will return `True` if the similarity of the names is below the given threshold (0.7). Now we know that the name values have a similarity between 0.5 and 0.7, which it can be considered a reject case, but instead of just rejecting it, we will also mark it ambiguous in order to be able to check these examples later. The third condition evaluates to `True` for examples with name similarities between 0.7 and 0.9. We want to keep these links and apply the fusion process, but the similarity score does not indicate an exact match, so we mark these links as ambiguous. Finally, in case none of the three conditions evaluate to `True`, the default validation action will apply which will accept the link (*name values should be almost identical with similarity above 0.9*).

4.1.3. Recommending POI link validation and fusion

Moving beyond user-defined rule specifications for link validation and fusion, FAGI v2.0 incorporates functionality for *automatic recommendation* of validation and fusion actions. This functionality is constantly being evaluated and improved using feedback from our industrial partners derived from real-world data integration jobs, with the aim to further automate the fusion of POI data, as well as to facilitate the quality assurance processes. In this subsection, we briefly describe the problem definition and formulation and an overview of the initial solutions we have developed. The specific methods we have introduced and developed to handle these tasks are thoroughly presented in Section 3.

4.1.3.1. Problem formulation

FAGI receives as input a pair of POI datasets, A and B, which contain POI entities accompanied with several properties, as well as a links file L, which contains a set of links connecting POIs from A with POIs from B.

Considering a POI *a* in A that is linked with a POI *b* in B, and the sets of properties {$P_{a,i}$} and {$P_{b,i}$} that describe them respectively, FAGI handles two tasks:

- **Decide validation action**. Decide whether POIs *a* and *b* actually correspond to the same real-world entity or they are wrongly interlinked. This can be formulated as a binary classification problem with output classes "accept" and "reject".
- **Decide fusion action**. Decide which is the most fitting fusion action for each pair *i* of matching properties ($P_{a,i}$, $P_{b,i}$) of the two POIs. This can be formulated as a multi-class classification problem, with the different fusion actions being considered as the classes of the problem.

Our goal is to develop the functionality that facilitates the accurate execution of the above two tasks, requiring the minimum effort by the user both in the selection of validation/fusion actions and in the post-processing, manual examination/validation of the fusion results.

4.1.3.2. Overview of solutions

Similarity-based link validation. The first task that needs to be handled by FAGI is link validation. Our first approach is to solve this problem through a similarity-based method, following a wide range of works in the literature. Specifically, our goal has been to design a POI specific meta-similarity function that can be used to accurately identify/distinguish between correctly and wrongly linked POIs. The method we implemented, called *FAGISimilarity*, comprises several string processing steps, a limited set of parameters-weights, an underlying, basic similarity function and a similarity threshold that can be identified at an initial fine-tuning stage. The main objective of FAGISimilarity is to identify and separately compare different parts of the names of two linked POIs, based on the rationale that (a) matching and non-matching parts of the names should be compared differently and (b) matching of frequent terms should be compared differently. FAGISimilarity can be characterized as a "meta-similarity" function, in the sense that it prescribes a series of operations to be performed, utilizing more basic similarity measures in several steps. In our method, we support the utilization of the following basic similarity measures: *Levenshtein, 2-Gram, Longest Common Subsequence, Jaro, Jaro-Winkler*.

Supervised link validation and fusion action recommendation. These two tasks are jointly solved by considering exactly the same learning process and only differentiating between either binary, or multi-class classification. This approach takes as input training data provided by experts-stakeholders. These data are comprised by pairs of linked POIs, along with their matching properties. The expert assigns, for each such training instance, a *link label* denoting whether it is a valid link or not, as well as a set of *fusion labels* denoting the proper fusion action per pair of matching properties. The set of training instances with the respective labels train two distinct classifiers in order to learn two models for the two tasks at hand: (i) classification of a new pair of linked POIs as "accept" or "reject"; (ii) Classification of each pair of matching properties to one of the available fusion actions in the system.

4.1.3.3. FAGI v2.0 support

Currently, the automatic validation and fusion recommendation functionality is integrated in FAGI v2.0 as an instrument *auxiliary* to the rule specification functionality mechanism. As such, the user performs validation and fusion actions by defining validation and fusion rule specifications and without advising the recommendation mechanism. However, FAGI runs the recommendation mechanism on each instance and

produces labels both for link validation and for fusion of individual properties. In case these recommendations disagree with the selected actions, the respective fusion results are marked as ambiguous (*in the same way described in Section 4.1.1.1*), so that the integrator can review them in a later stage. During the second development iteration of FAGI, and while we further fine-tune our algorithms with respect to recommendation accuracy, the role of FAGI's learning/recommendation mechanism will be upgraded, so as to guide and further automate the fusion process, further reducing the need for manual validation of fusion results.

4.1.4. Quality indicators and statistics

FAGI v2.0 supports the extraction of an extended set of quality indicators and statistics, both at the beginning, and the end of the fusion process. The user is able to review several statistics on the input, linked POI datasets, before performing fusion on them (*pre-fusion statistics*), as well as on the output, fused data (post-fusion statistics). The goal of the former is to allow the integrator to obtain an overview of the data at hand, which may help her properly define and configure the validation/fusion rules. The goal of the latter is to assist the user in the examination/validation of the fusion results, and potentially guide her into re-configuring and re-executing the fusion process.

4.1.4.1. Pre-fusion statistics

FAGI v2.0 supports the extraction of the following statistics/indicators:

- Statistics on individual input datasets:
    - Number of POI entities in each input dataset.
    - Total number of triples in each input dataset, i.e., total number of properties for all POIs.
    - Total numbers of empty and non-empty triples in each input dataset.
    - Average number of properties per POI in each input dataset.
    - Average number of empty and non-empty properties in each input dataset.
    - Average number of categories(tags) per POI in each input dataset.
    - Total number of POIs that have a specific property in each input dataset.
    - Number of empty and non-empty values a specific property in each input dataset.
- Statistics related to linked POIs of the input datasets:
    - Ratio of linked POIs to total number of POIs in each input dataset.
    - Total number of triples in each input dataset (*i.e., total number of properties for all POIs*), corresponding only to linked POIs.
    - Total numbers of empty and non-empty triples in each input dataset, corresponding only to linked POIs.
    - Average number of properties per POI in each input dataset, corresponding only to linked POIs.
    - Average number of empty and non-empty properties in each input dataset, corresponding only to linked POIs.

- Total number of POIs that have a specific property in each input dataset, regarding only linked POIs.
- Total numbers of empty and non-empty values for a specific property in each input dataset, regarding only linked POIs.
- Average number of categories(tags) per POI in each input dataset, regarding only linked POIs.
- Number of POI name property values from dataset A that are longer (longer literals) than the names of the corresponding (linked) POIs from dataset B (also the inverse indicator).
- Number of POI phone property values from dataset A that are longer than the names of the corresponding (linked) POIs from dataset B (*longer phone strings imply more proper phone format, e.g., containment of full country/exit codes*) (also the inverse indicator).
- Number of fully matching address streets between linked POIs in the two datasets.
- Number of fully matching address numbers between linked POIs in the two datasets.

- Frequent terms statistics. For several important POI properties (*e.g., name, address, categories*) a list of frequent terms on each dataset is produced.

4.1.4.2. Post-fusion statistics

FAGI v2.0 supports the extraction of the following statistics/indicators:

- Number of fused POIs vs. initial links, i.e., the number of POI links that were not rejected by FAGI and were actually considered for fusion of their attributes.
- Number of rejected POI links vs. initial links, i.e., the number of POI links that were eventually rejected by FAGI.
- Number of fusion actions that: Kept left value; Kept right value; Concatenated left and right value and kept as one; Kept both values as separate properties; Kept longest value
- For each fusion rule that was defined in the fusion specification, the number of times it was executed and produced a fused POI.
- For each link validation rule that was defined in the validation specification, the number of times it was executed.
- Number/percentage of fused POIs that were marked as ambiguous (and thus require further examination/validation) vs. the number of initial links.
- Number/percentage of rejected POIs that were marked as ambiguous (and thus require further examination/validation) vs. the number of initial links.

4.1.5. Dataset output modes

FAGI v2.0 supports the modes enumerated in the following list, for outputting fused POI datasets. Note that the input of FAGI v2.0 remains fixed and is composed from two source POI datasets and a set of links that link POIs between the two datasets. Further, in all but one of the output modes, the user is required to select a "base" dataset, from the two input POI datasets (*the remaining input dataset is then denoted as*

"secondary"). In most cases, this base dataset will be used as a basis for outputting the final, fused POI dataset.

1. **Handle only linked POIs – Include base dataset**. The pairs of linked POIs are fused into unified descriptions, and the resulting fused POI entities are written on the fused dataset. All unlinked POIs from the base dataset are also written on the fused dataset. All unlinked POIs from the secondary input dataset are discarded.
2. **Handle only linked POIs – Use a new dataset**. Only the results of the fusion process are written to the fused dataset. The new dataset contains unified descriptions only for pairs of POIs that where linked. Unlinked POIs from both input datasets are discarded.
3. **Handle both linked and unlinked POIs – Produce a single dataset**. The pairs of linked POIs are fused into unified descriptions, and the resulting fused POI entities are written on the fused dataset. All unlinked POIs from the base dataset are also written on the fused dataset. In contrast to the first mode, the unlinked triples of the secondary dataset are also written to the fused dataset. This way, the final, fused dataset contains fused POIs as prescribed from the links between the initial two input datasets, as well as **all** unlinked POIs from both input datasets.
4. **Handle both linked and unlinked POIs – Produce two datasets**. In this mode, two output datasets are produced. The first output dataset contains exactly the same data as the output dataset of mode 1. The second output dataset is produced by subtracting from the secondary dataset all the POIs (*with all their properties*) that participated in the fusion process. This way, the first output dataset represents the result of the fusion, while the second output dataset contains all the POIs that are not eventually included in the first output dataset.

4.2. Architecture

FAGI v2.0 is intended to be used for the efficient and accurate fusion of large volumes of linked POI entities. It can be executed as an autonomous service, or as part of an integration workflow within the SLIPO Workbench. FAGI comprises several modules that isolate different functionalities and allow their modular extension. Figure 6 provides an overview of FAGI's architecture.

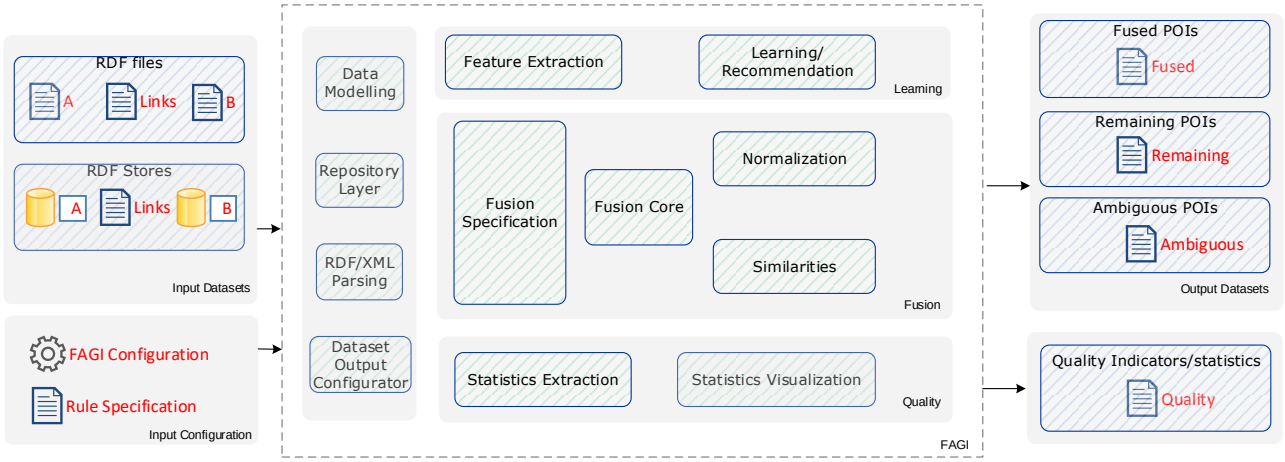


Figure 6: FAGI v2.0 Architecture

Based on the above architecture, we next describe in detail the input/output of FAGI, as well as the major components and modules it consists of.

4.2.1. Input and output

FAGI v2.0 accepts as input two POI datasets (A and B) in RDF format and a file containing *sameAs* links that connect POIs between the two datasets. Two input source types are supported, that essentially cover most usual input cases: RDF files and SPARQL endpoints. Regarding RDF files, the following RDF formats are supported:

- N-Triples (NT)
- Turtle (TTL)
- RDF/XML (RDF)
- RDF/XML (OWL)
- JSON-LD (JSONLD)
- RDF/JSON (RJ)
- TriG (TRIG)
- N-Quads (NQ)
- TriX (TRIX)

Additionally, FAGI receives as input an execution configuration and a rule specification containing link validation and fusion action rules. The structure of the configuration and rule specification files is described in Section 5.2. Also, examples of validation and fusion rule specifications are described in Figure 4 and Figure 5.

FAGI produces four different types of output files, depending on the configuration and the selected dataset output mode (see Section 4.1.5).

- **Fused POIs dataset**. It is the core output of FAGI and contains the fused POIs, as produced by the executed validation and fusion rule specifications. Depending on the configuration, the fused POI dataset may be produced in three variations:
    - **Plain**. In this version, the fused dataset does not include any additional POIs, apart from the fused POIs.
    - **Fused and unlinked from base**. In this version, the fused dataset also includes the remaining, unlinked POIs from one of the two input POI datasets, that serves as the "base" dataset.
    - **Fused and all unlinked**. In this version, the fused dataset also includes the remaining, unlinked POIs from both input POI datasets.
- **Remaining POIs dataset**. This dataset is optionally produced only if it is prescribed in the FAGI configuration by the user and mainly regards the case where the fused POIs dataset also contains the unlinked POIs from the base input POI dataset. In this scenario, the remaining, unlinked POIs

from the secondary input POI dataset comprise the remaining POIs dataset. That is, this output dataset contains all the POIs from the secondary input POI dataset, except for the ones that where eventually fused.

- **Ambiguous POIs dataset**. This dataset is created in order to retain ambiguous POIs from the initial datasets, in case they are required for additional processing, after the examination/validation of the fused POIs. We note that the whole set of POI properties are retained, so that all information is available at a later integration stage.
- **Quality indicators/statistics dataset**. This dataset contains all the statistics and quality indicators extracted by FAGI, as prescribed in Section 4.1.4.

4.2.2. Core modules

FAGI v2.0 currently consists of four (4) high level components:

- **I/O Component**. This component consists of three main sub-modules. The module for reading input data and writing the output results, the module for parsing and validating the configuration and input files, and the repository layer. The repository layer serves as an in-memory object collection and encapsulates the functionality for querying and retrieving any necessary data during the fusion process.
- **Quality Component**. This component implements all the functionality for extracting statistics and indicators from both the input and the output data of FAGI as well as throughout the validation and fusion process.
- **Fusion Component**. This component handles the core, as well as auxiliary functionality related to fusion. This includes the definition and processing of rule specifications, the evaluation of conditions and expressions, the application of fusion actions, similarity calculations, normalization and comparison of property values, as well as the link validation process.
- **Learning Component**. This component is responsible for the learning mechanisms that allow training on previous actions and recommending fusion and validation actions for new pairs of linked POIs.

In what follows, the basic modules of FAGI v2.0 are briefly presented:

- **Fusion core**. It is the module that gathers all the core functionality w.r.t. fusing POIs. In particular, it implements all available fusion and link validation actions, as well as all the functions that implement conditions on POI properties. The core is responsible for carrying out all the fusion processes based on the dataset output mode and producing the final output results.
- **Fusion Specification**. This module implements the functionality for defining rule specifications for link validation and fusion. The fusion specification supplies all necessary methods for validating input (either in XML format, or in the format produced by FAGI's software interface, which is consequently transformed to FAGI's XML syntax for rule specifications) and manages the process of parsing the XML configuration files as well as mapping and modelling them into in-memory data structures suitable for guiding the validation and fusion process.

- **Normalization**. This module gathers all pre-processing functionality that is executed on the values of several POI property types, with the aim to clean the data to some extent, as well as to align them in terms of format. It consists of several submodules, each one responsible for generic or more specific data types, depending on the objective of the specification.
- **Similarities**. This module implements a series of similarity functions, rules and thresholds that allow the comparison of POI property values of several types (e.g., addresses, names, phones). The functions that are implemented in this module are utilized in several others, such as in Fusion Specification, Normalization and Learning/Recommendation.
- **Feature Extraction**. This module realizes a series of functions for extracting features that represent pairs of linked POIs. These features include characteristics of property values of the POIs, as well as relations between these property values.
- **Learning/Recommendation**. This module implements several learning (classification) algorithms that are able to recommend: (a) whether a pair of linked POIs should be accepted (fused) or rejected; (b) which is the proper fusion action for a pair of matching properties between two linked POIs.
- **Statistics Extraction**. This module extracts statistics from the input and output data of FAGI. These statistics are destined both for exploration by the end user and for utilization in other modules of FAGI (e.g., Feature Extraction, Similarities).
- **RDF Parsing**. This module handles parsing input RDF data (either from RDF files, or from a SPARQL endpoint) and loading them into RDF models in memory. The supported RDF formats are described in Section 4.2.1.
- **Data Modelling**. This module creates models and mappings for the input POIs and their links and encapsulates all the necessary functionality for accessing and handling the specification logic in order to support the actual validation and fusion of POI data.
- **Repository Layer**. Essentially, it provides an abstraction of data, so that the application can work with a simple abstraction that has an interface approximating that of a collection. Adding, removing, updating, and selecting items from this collection is done through a series of straightforward methods, without the need to deal with the data sources, e.g., files, SPARQL endpoints or in memory graphs.
- **Dataset Output Configurator**. This module handles the formatting of the output, fused POIs into a proper RDF format, according to SLIPO's ontology. Further, it realizes several options for the number and types of output datasets, as described in Sections 4.1.5 and 4.2.1.

4.2.3. Partitioning scheme

One of our main goals in the SLIPO project is to produce a commercial-level fusion software that can scale to tens of millions of POIs and efficiently execute batch fusion of large POI datasets. As a first step towards this direction, FAGI v2.0 implements a partitioning scheme for dividing the input POI datasets into a configurable number of subsets and distributing their processes into multiple parallel instances of FAGI.

The goal is to divide the two input POI datasets A and B into ($N_A \times N_B$) partitions that match a corresponding partition of the set of links between POIs. Specifically, we want to end up with a number of directories (*partition-directories*) of specific size (*calculated so that a single moderately size in terms of memory FAGI*

instance can handle them). Each partition-directory should contain a set of FAGI input files (*dataset-partitions*): (i) a partition of the initial links file (*links-sublist*), (ii) a partition of dataset A containing all the POIs contained in the *links-sublist,* and (iii) a partition of dataset B containing all the POIs contained in the *links-sublist*.

The partitioning process consists in four main steps:

(a) **Partitioning the links file**. This step comes before the actual dataset partitioning and is straightforward, since the links file consists of independent triples (n-triples format) in the form of "<POI_a> <owl:sameAs> <POI_b>", where *a* and *b* are POIs belonging to datasets A and B respectively. The number of the initial partitions of the links file is calculated with a rough estimation of the initial file size and the available memory of each *FAGI-partitioner*. Then, each of these *links-lists* should be able to be managed by each *FAGI-partinioner*.

(b) **Creating *links-sublists* and mapping them to *partition-directories***. Each of the *links-lists* are loaded into different instances of *FAGI-partinioners* that are now able to load them in memory. Then, each *FAGI-partitioner* is responsible for further dividing the *links-list* in order to produce groups of subsets of the input datasets (dataset A, dataset B, links file) in a way that each group can fit in the provided memory of the machine that runs a *FAGI-instance*. Following an estimation of the partition size, the *FAGI-partitioner* divides the *links-list* into as many *links-sublists* as required and produces a *partition-directory* for each *links-sublist*, while serializing each *links-sublist* to an RDF file (n-triples format) inside that directory.

(c) **Mapping POI URIs of each link to a *dataset-partition* filename inside a *partition-directory*.** This process creates three mappings for each *links-sublist*: (i) the first one maps the URIs from dataset A to a corresponding *dataset-partition* filename using the *links-sublist* hashcode in order to keep track of which URIs belong to which *dataset-partition* and (ii) similarly, the second one maps the URIs from the dataset B to the corresponding *dataset-partition* filename. These mappings are constructed for later use in the actual partitioning process. This process uses multimaps, as there is no guarantee that the URIs are unique between links (a URI from the dataset A could be linked with several different URIs from dataset B).

(d) **Partitioning of the source datasets**. Finally, the *FAGI-partitioner* is ready to partition the input RDF data. This step creates two threads, one for each input RDF file and the corresponding mappings from (c). Each thread starts streaming the input RDF file and, by utilizing the URI mappings, is able to match (and write) every triple in its corresponding *dataset-partition*.

The result of this process is a group of independent partitions of linked RDF data that are ready to be passed in multiple FAGI instances separately. At the end of the fusion process, the output of all FAGI instances are combined to a single (or more, depending on the fusion mode) output result.

4.3. Libraries and Frameworks

FAGI has dependencies to the following open-source tools/libraries:

- Apache Jena[2]: A Java framework for building Semantic Web applications.
- Java Topology Suite[3]: An API of 2D spatial predicates and functions conforming to the OGC Simple Features Specification for SQL.
- Google Guava[4]: A set of libraries that includes collection types (such as multimap and multiset), immutable collections, a graph library, functional types, an in-memory cache, and APIs/utilities for concurrency, I/O, hashing, primitives, reflection and string processing.
- Apache Commons Text[5]: A library focused on algorithms working on strings.
- Apache Commons Lang[6]: Provides a host of helper utilities for the java.lang API, notably String manipulation methods, basic numerical methods, object reflection, concurrency, creation and serialization and System properties. Additionally it contains basic enhancements to java.util.Date and a series of utilities dedicated to help with building methods, such as hashCode, toString and equals.
- Apache Lucene[7]: A high-performance, full-featured text search engine library written entirely in Java. It is a technology suitable for nearly any application that requires full-text search, especially cross-platform.
- Google code json-simple[8]: A simple Java toolkit for JSON. It is used to encode or decode JSON text.
- Apache Log4j 2[9]: A Java-based logging utility.

Additionally, the UI component of FAGI is a Spring Boot application and uses the following libraries:

- Spring Boot[10]: Starter for building web, including RESTful, applications using Spring MVC. It uses Tomcat as the default embedded container.
- Spring Boot Developer Tools[11]: Additional set of tools that facilitate the application development experience. The spring-boot-devtools module can be included in any project to provide additional development-time features.
- Google Code Gson[12]: A Java serialization/deserialization library to convert Java Objects into JSON and back.
- JDOM 2[13]: A complete, Java-based solution for accessing, manipulating, and outputting XML data.

---

[2] https://jena.apache.org/

[3] https://github.com/locationtech/jts

[4] https://github.com/google/guava

[5] https://commons.apache.org/proper/commons-text/

[6] https://commons.apache.org/proper/commons-lang/

[7] https://lucene.apache.org/

[8] https://code.google.com/archive/p/json-simple/

[9] https://logging.apache.org/log4j/2.x/

[10] https://spring.io/projects/spring-boot

[11] https://mvnrepository.com/artifact/org.springframework.boot/spring-boot-devtools/1.3.0.RELEASE

[12] https://github.com/google/gson

[13] https://mvnrepository.com/artifact/org.jdom/jdom2/2.0.6

4.4. License

FAGI is an open source software and is available from GitHub at [FAGI] (the core software) and at [FAGI-web] (the UI layer). It can be redistributed and/or modified under the terms of the Apache version 2.0 License as published by the Apache Software Foundation.

4.5. Documentation

A JavaDoc API documentation has been prepared in HTML format is publicly available at [FAGI-api]. Full documentation for the FAGI software is provided in [FAGI-doc].

5. Usage Manual

In this section, we provide the usage manual for FAGI v2.0. First, we give details on building the application from the Java source code, along with information on its dependencies. Next, we provide instructions on constructing link validation and fusion action rule specifications, and for configuring the execution of FAGI. Finally, we present a short example demonstrating the configuration and execution of FAGI.

5.1. Building Installation

FAGI v2.0 is available as an open source software, with its source code as well as indicative configurations available for download at [FAGI][FAGI-web]. Java SDK 1.8 (or later) as well as Maven 3.3.3 (or later) [MAVEN] must be installed and properly configured in order to compile and execute FAGI. The `pom.xml` file contains the project's configuration in Maven and has been successfully tested in both Microsoft Windows and Linux environments. The following building instructions assume that Git is also installed.

5.1.1. Command line version

In order to build the command line version from source, first the master branch of FAGI must be cloned to a preferred location by running:

```
git clone -b master --single-branch https://github.com/SLIPO-EU/FAGI.git fagi
```

It is recommended to use the `--single-branch` parameter to save some time and avoid pulling the whole history of the project.

Then, from the root directory of the project (fagi) the following command needs to be executed:

```
mvn clean install
```

After a successful installation, a target directory should have been created containing the fagi-2.0-SNAPSHOT.jar (version depending on POM configuration).

In order to deploy FAGI, the following command needs to be executed under the target directory:

```
java -jar fagi-1.2.3-SNAPSHOT.jar -spec /path/to/conf.xml
```

where “conf.xml” is the FAGI v2.0 configuration file (described in Section 5.2.1).

5.1.2. Web interface version

Optionally, FAGI provides a web interface which helps the user build complicated rule specifications without having to manually define rules in XML files. The web interface is a Spring Boot application which makes it very easy to install and use. The web interface installation assumes that the command line version is already installed and uses the command-line version as a library.

Similarly, the FAGI-web repository needs to be cloned to a preferred location on your system by running:

```
git clone https://github.com/SLIPO-EU/fagi-web.git fagi-web
```

In order to build the project, the following command needs to be executed from the root directory of the project (fagi-web):

```
mvn clean install
```

After a successful installation, the project can be easily started by running the following command in the same directory:

```
mvn spring-boot:run
```

Upon the application has been initialized and started, the web interface is accessible through a browser at by visiting

```
http://localhost:8080
```

5.2. Configuration Settings

FAGI requires two XML files as configuration input. These files contain some basic input/output information, as well as the fusion modes and the rule specification. In this section, we describe the structure and syntax of these files and we provide some basic examples.

5.2.1. Configuration specification

The first of these files is provided through the command line with the "-co" parameter followed by the path of the configuration file. The configuration specification follows an XML syntax and holds general configuration for the fusion process, which is filled with text values between an opening and a closing tag. A template of this file looks like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<specification>
	<inputFormat></inputFormat>
	<outputFormat></outputFormat>
	<locale></locale>
	<similarity></similarity>
	<rules></rules>
	<left>
		<id></id>
		<endpoint></endpoint>
		<file></file>
		<categories></categories>
	</left>
	<right>
		<id></id>
		<endpoint></endpoint>
		<file></file>
		<categories></categories>
	</right>
	<links>
		<id>links</id>
		<endpoint></endpoint>
		<file></file>
	</links>
	<target>
		<id>fused</id>
		<mode></mode>
```

```
            <outputDir></outputDir>
            <fused></fused>
            <remaining></remaining>
            <ambiguous></ambiguous>
            <statistics></statistics>
        </target>
</specification>
```

The *inputFormat* refers to the RDF format of the input dataset and the *outputFormat* holds the value of the desired output format. The valid RDF formats are:

- N-Triples (NT)
- Turtle (TTL)
- RDF/XML (RDF)
- RDF/XML (OWL)
- JSON-LD (JSONLD)
- RDF/JSON (RJ)
- TriG (TRIG)
- N-Quads (NQ)
- TriX (TRIX)

In order to fill the *inputFormat* and *outputFormat*, the values in the corresponding parentheses are used within the file.

The *locale* is an optional parameter, in case a dataset contains entities from regions with different locales, but it is strongly recommended to choose one when applicable, because it is used on several steps of the normalization process. The available locales are:

- EN
- EN-GB
- EN-US
- DE
- DE-DE
- DE-AT
- EL

The *similarity* parameter is used as a part of the customized similarity function we implement within FAGI. The available values (case-insensitive) are the following (default is *JaroWinkler*):

- levenshtein
- 2Gram
- longestcommonsubsequence

- jaro
- jarowinkler

The *rules* field expects the absolute path of the rule specification XML file, which is discussed in Section 5.2.2.

The *left*, *right*, *links* and *target* parameters refer to the source and target datasets. Each of these XML tags contain additional tags that describe each of the datasets. Specifically:

- *id*. An ID to identify the dataset.
- *file*. The filepath of the dataset.
- *endpoint* (optional parameter). Instead of using files, add a SPARQL endpoint and leave the file tag empty.
- *categories* (optional parameter). It is used to extract statistics about the categories of the entities. It requires the filepath of a file in N-Triples format that contains the categorization.

The *mode* parameter is used in order to specify the dataset output mode. The supported parameter values are the following:

- *aa_mode*: Only linked triples are handled – include base dataset. Fused triples along with unlinked triples only from dataset A are written in the output, fused dataset.
- *bb_mode*: Only linked triples are handled – include base dataset. Fused triples along with unlinked triples only from dataset B are written in the output, fused dataset.
- *ab_mode*: All triples are handled: Fused triples along with unlinked triples from both datasets are written in the output, fused dataset, using dataset A as base dataset.
- *ba_mode*: All triples are handled: Fused triples along with unlinked triples from both datasets are written in the output, fused dataset, using dataset B as base dataset.
- *a_mode*: All triples are handled. Fused triples along with unlinked triples only from dataset A are written in the output, fused dataset. Triples corresponding to fused POIs are removed from dataset B, which only maintains the remaining, unlinked triples.
- *b_mode*: All triples are handled. Fused triples along with unlinked triples only from dataset B are written in the output, fused dataset. Triples corresponding to fused POIs are removed from dataset A, which only maintains the remaining, unlinked triples.
- *l_mode*: Only linked triples are handled: Only fused triples are written in the output, fused dataset.

The *outputDir* parameter holds the directory path under which all produced files will be written. The produced files are described below:

- *fused* (optional parameter). Specifies the output filepath of the fused dataset (based on fusion mode). If no value is specified the default name will be "fused.nt" under the output directory defined above.

- *remaining* (optional parameter). Specifies the output filepath of the non-fused dataset, containing the remaining, unlinked triples (based on fusion mode). If no value is specified the default name will be "remaining.nt" under the output directory defined above.
- *ambiguous* (optional parameter). Specifies the output filepath of the dataset containing ambiguous linked entities. If no value is specified the default name will be "ambiguous.nt" under the output directory defined above.
- *statistics* (optional parameter). Specifies the path of the statistics file. By default, a file with name "statistics.txt" will be written under the output directory defined above.

5.2.2. Rule Specification

The second configuration input for FAGI is the rule specification file. This file holds all the rules that the validation and fusion processes are to follow and requires that certain text values are filled between an opening and a closing tag, similarly to the configuration specification file. The file starts with the root element <*rules*> and each fusion rule on a property is set using a <*rule*> element as a child of the root tag. Each <*rule*> element consists of the following main childs: <propertyA>, <propertyB>, <externalProperty>, <actionRuleSet>, <defaultAction>.

- <propertyA> and <propertyB> define the two matching RDF properties that the rule will apply the fusion action upon.
- <externalProperty> is used to define properties that are utilized inside condition functions. The fusion action does not affect the value of this property. The external property requires an id attribute as a parameter in the XML and the id must start with the letter *a* or *b* that refers to the corresponding value (from the corresponding input POI dataset) and followed by an incrementing integer for each different property used in the same rule.
- <actionRuleSet> element: This element consists of one or more <actionRule> child elements. Each is a pair of a condition expression and a fusion action, namely `(condition expression, fusion action)`, denotes by tags <condition> and <action> respectively. If the condition expression evaluates to `True`, then the respective fusion action of the specific action rule will be applied and all the next action rules will be ignored.
- <defaultAction> is the default fusion action to apply if no condition from the is met.

The condition expression (<condition>) along with the fusion action (<action>) are the most essential part of the configuration of the fusion process. In order to construct a condition expression, we assemble a group of logical operations that contain condition functions to apply on the RDF properties defined above. We can define a logical operation by using the <expression> tag as a child of a condition. Then, inside the expression we can put together a combination of <and>, <or> and <not> operations. As operands we can use <function> elements containing a condition function or a nested expression containing more logical operations. The depth of the nested expressions supported currently is 2 levels.

Apart from fusion rules which are defined using the <rule> tag, similar functionality is supported for the definition of validation rules using the <validationRule> tag. With a validation rule we can accept/reject and/or mark a link as ambiguous in the model. The validation rules follow the exact same logic described

above with the only difference being that the fusion actions are replaced with the validation actions. Examples of rule configurations are provided in Sections 4.1.1.3, 4.1.2.3 and 5.3

5.3. Demonstration

Next, we demonstrate the usage of FAGI v2.0 through an exemplary execution on real world POI datasets. FAGI aims at facilitating both expert and non-expert stakeholders of the POI value chain into fusing batch amounts of linked POIs. The core usage of the software consists in a very simple workflow: (i) the user provides a minimal input configuration for the execution of FAGI; (ii) the user provides link validation and fusion rules within the rule specification of FAGI; (iii) the user deploys FAGI with input the two aforementioned configuration files and obtains the fusion results, which might consist in four different files, depending on the selected dataset output mode, as prescribed in Sections 4.2.1 and 5.2.1. In what follows, this workflow is demonstrated in detail.

5.3.1. FAGI input configuration

The first step for the user is to provide the configuration of FAGI, following the instructions of Section 5.2.1. As an example, the following configuration can be produced:

```
<?xml version="1.0" encoding="UTF-8"?>
<specification>
	<inputFormat>NT</inputFormat>
	<outputFormat>NT</outputFormat>
	<locale>EN</locale>
	<similarity>jarowinkler</similarity>
	<rules>./rulespecs.xml</rules>
	<left>
		<id>datasetA</id>
		<endpoint></endpoint>
		<file>G:\DATA\RDF\TomTom_MultiNet_Austria_RDF_v1.4\TomTom_mnpoi_Austria.nt</file>
		<categories></categories>
	</left>
	<right>
		<id>datasetB</id>
		<endpoint></endpoint>
		<file>G:\DATA\RDF\Herold_RDF_v.0.4\public.herold.nt</file>
		<categories></categories>
	</right>
	<links>
		<id>links</id>
		<endpoint></endpoint>
		<file>G:\DATA\RDF\links.nt</file>
	</links>
	<target>
		<id>fused</id>
		<mode>aa_mode</mode>
		<outputDir> G:\DATA\RDF\output</outputDir>
		<fused></fused>
		<remaining></remaining>
		<ambiguous></ambiguous>
		<statistics></statistics>
	</target>
</specification>
```

In this example, we have set the paths of the RDF input files, the path for the rule specification file, the output directory of the results, the locale of the input datasets and the dataset output mode. The latter prescribes that only the linked/fused POIs along with the unlinked POIs from the left datasets are maintained in the fused, output dataset. This configuration can either be provided directly in the configuration specification file, or be filled in through the graphical user interface of FAGI.

5.3.2. FAGI validation and fusion rule specification

The second step is to define validation and fusion rules in the rule specification XML file. In this scenario, we are interested in validating links by checking similarities on their name property values. We wish to define two validation conditions with the following actions: (i) reject links with POI name similarities (normalized) less than 0.5, and (ii) accept and mark as ambiguous all links that have POI name similarities between [0.5,0.7) and their phone number values are the same. The priority of the validation rules (according to their order of definition in the file) will ensure that validation action (ii) will be applied only to POIs with name similarities above 0.5. The prescribed validation specification is presented below:

```
<?xml version="1.0" encoding="UTF-8"?>
<rules>
  <validationRule>
    <defaultAction>accept</defaultAction>
    <actionRuleSet>
      <actionRule>
        <action>reject</action>
        <condition>
          <expression>
            <not>
              <function>isSameCustomNormalize(a0, b0, 0.5)</function>
            </not>
          </expression>
        </condition>
      </actionRule>
      <actionRule>
        <action>accept-mark-ambiguous</action>
        <condition>
          <expression>
            <and>
                        <function>isSamePhoneNumber(a0, b0)</function>
              <expression>
                <not>
                  <function>isSameCustomNormalize(a1, b1, 0.7)</function>
                </not>
              </expression>
            </and>
          </expression>
        </condition>
      </actionRule>
    </actionRuleSet>
    <externalProperty id="a0">http://slipo.eu/def#phone http://slipo.eu/def#contactValue
    </externalProperty>
    <externalProperty id="b0">http://slipo.eu/def#phone http://slipo.eu/def#contactValue
    </externalProperty>
    <externalProperty id="a1">http://slipo.eu/def#name http://slipo.eu/def#nameValue
    </externalProperty>
    <externalProperty id="b1">http://slipo.eu/def#name http://slipo.eu/def#nameValue
    </externalProperty>
  </validationRule>
```

The above validation specification structure can be provided directly in the configuration specification file; however, the syntax may become quite complicated depending on the rules we define. For this purpose, we have developed a web interface for constructing more complicated rule specifications. The above XML file can be easily constructed using the UI and defining rules, through the selection of properties, conditions functions, condition expression operators and actions, as seen in Figure 7:

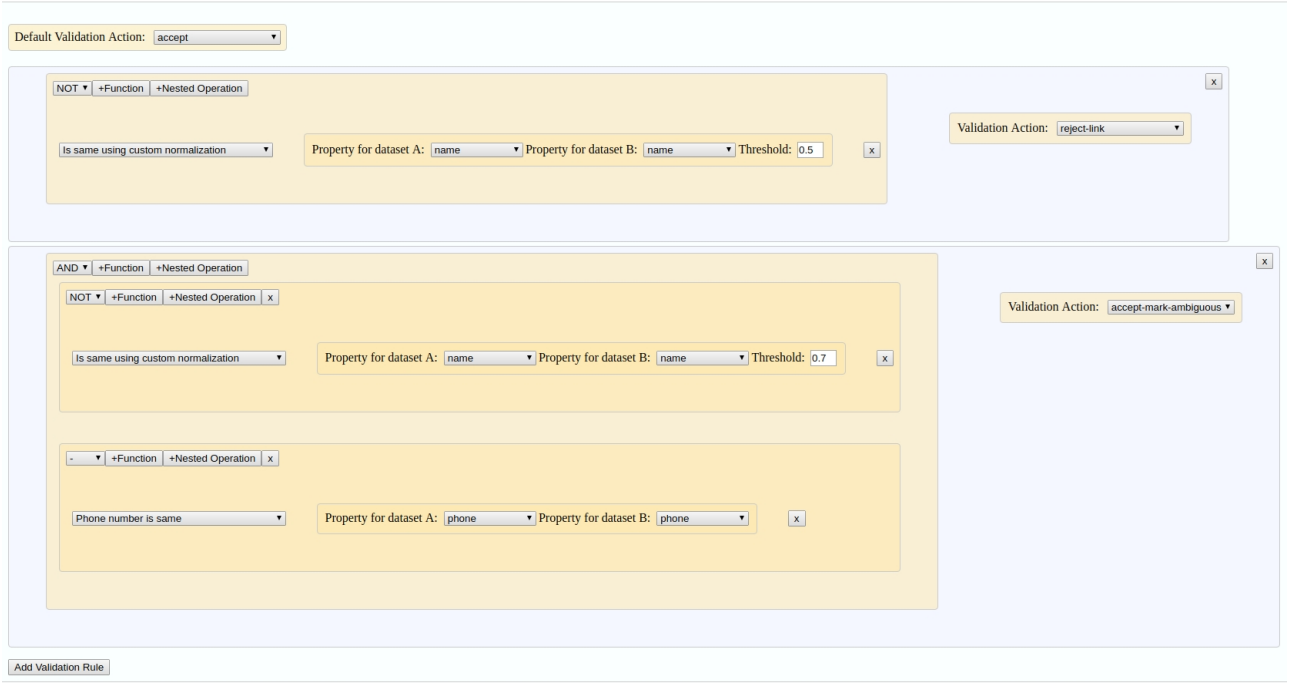


Figure 7: UI for validation rule specification

Regarding fusion, we wish to define a rule prescribing the fusion of the POI name property values, as well as a rule prescribing the fusion of phone number values. All other properties should be kept from the left dataset, thus we define "keep-left" as the default dataset-level action. For the first action rule (fusing names), we want to *keep the longer value* when the POI name values are at least 0.8 similar, their address's streets are almost identical (0.9 and larger) and their street numbers are equal. If this first action rule is not met, we define a secondary action rule that does not check the address values of the POIs, but instead it *keeps the left value* if the names are almost exactly the same (using normalization) and the left value does not contain an abbreviation. For all the remaining cases (when none of the defined action rules apply), we *keep both* name values (default action).

The above rule is realized in the XML specification syntax as presented in the listing below. In this, all the external properties that are used in the condition functions are defined at the end of the rule. Each of the two action rules includes a different fusion action and a different condition expression according to what was described in the previous paragraph. The rule also includes a default fusion action, in case no action rule is satisfied.

```
<rule>
    <propertyA>http://slipo.eu/def#name http://slipo.eu/def#nameValue</propertyA>
    <propertyB>http://slipo.eu/def#name http://slipo.eu/def#nameValue</propertyB>
    <defaultAction>keep-both</defaultAction>
    <actionRuleSet>
      <actionRule>
        <action>keep-left</action>
```

```
      <condition>
        <expression>
          <and>
            <function>isSameCustomNormalize(a2, b2, 0.8)</function>
            <function>isSameCustomNormalize(a3, b3, 0.9)</function>
            <function>isSameCustomNormalize(a4, b4, 1.0)</function>
          </and>
        </expression>
      </condition>
    </actionRule>
    <actionRule>
      <action>keep-right</action>
      <condition>
        <expression>
          <and>
                      <function>isSameCustomNormalize(a5, b5, 0.9)</function>
            <expression>
              <not>
                <function>isLiteralAbbreviation(b2)</function>
              </not>
            </expression>
          </and>
        </expression>
      </condition>
    </actionRule>
  </actionRuleSet>
  <externalProperty id="a2">http://slipo.eu/def#name http://slipo.eu/def#nameValue
  </externalProperty>
  <externalProperty id="b2">http://slipo.eu/def#name http://slipo.eu/def#nameValue
  </externalProperty>
  <externalProperty id="a3">http://slipo.eu/def#address http://slipo.eu/def#street
  </externalProperty>
  <externalProperty id="b3">http://slipo.eu/def#address http://slipo.eu/def#street
  </externalProperty>
  <externalProperty id="a4">http://slipo.eu/def#address http://slipo.eu/def#number
  </externalProperty>
  <externalProperty id="b4">http://slipo.eu/def#address http://slipo.eu/def#number
  </externalProperty>
  <externalProperty id="a5">http://slipo.eu/def#name http://slipo.eu/def#nameValue
  </externalProperty>
  <externalProperty id="b5">http://slipo.eu/def#name http://slipo.eu/def#nameValue
  </externalProperty>
</rule>
```

Similarly to the validation specifications, the user may use the graphical web interface of FAGI to easily construct the fusion rule of the above listing. Specifically, the user only needs to select the pair of matching properties and a default fusion action, and then start the construction the fusion action rules by selecting condition functions, operators and fusion actions from drop-down lists, that contain all supported functionality of FAGI. Figure 8 and Figure 9 (one for each of the two action rules) present the equivalent rule specification of the above listing, constructed through the web interface.

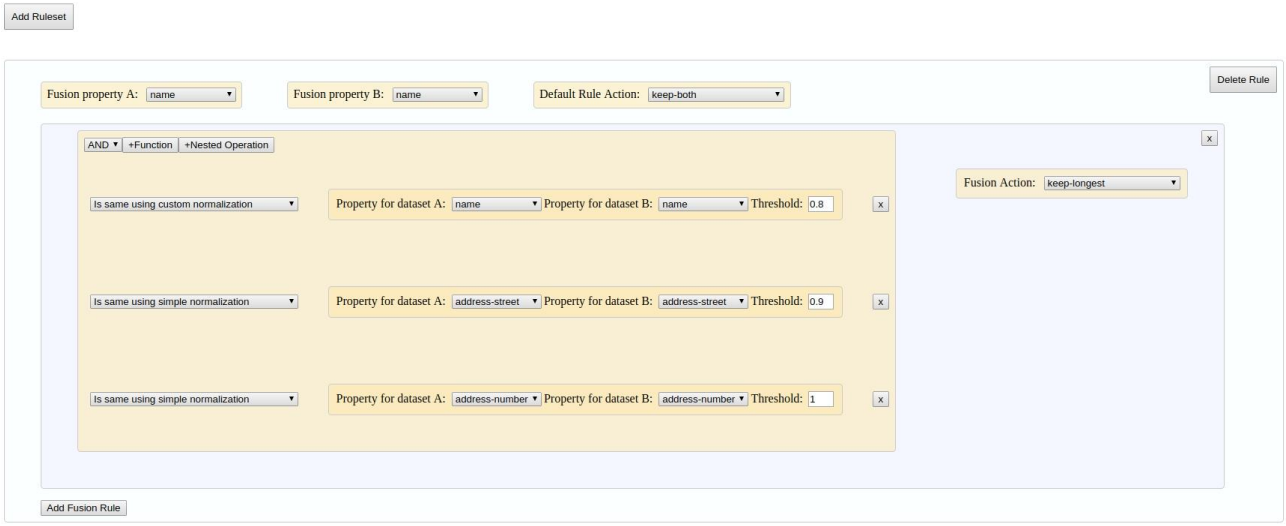


Figure 8: UI for fusion rule specification on POI names (first action rule)

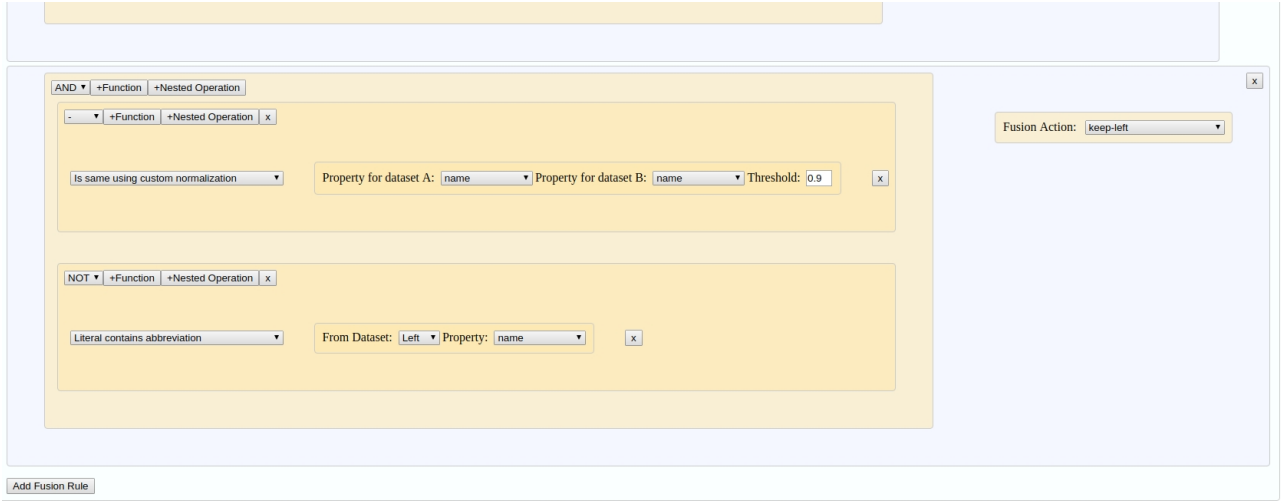


Figure 9: UI for fusion rule specification on POI names (second action rule)

Regarding the fusion of phone number values, we wish to *keep both* phone numbers as a default action. However, in case the phones are the same and their respective POI names are quite similar we only *keep the left* phone number value. This specification can be defined as follows in the rule specification file:

```
<rule>
  <propertyA>http://slipo.eu/def#phone http://slipo.eu/def#contactValue</propertyA>
  <propertyB>http://slipo.eu/def#phone http://slipo.eu/def#contactValue</propertyB>
  <defaultAction>keep-both</defaultAction>
  <actionRuleSet>
    <actionRule>
      <action>keep-left</action>
      <condition>
        <expression>
          <and>
            <function>isSameSimpleNormalize(a6, b6, 0.9)</function>
            <function>isSamePhoneNumber(a7, b7)</function>
          </and>
        </expression>
      </condition>
    </actionRule>
```

```
    </actionRuleSet>
    <externalProperty id="a6">http://slipo.eu/def#name http://slipo.eu/def#nameValue
    </externalProperty>
    <externalProperty id="b6">http://slipo.eu/def#name http://slipo.eu/def#nameValue
    </externalProperty>
    <externalProperty id="a7">http://slipo.eu/def#phone http://slipo.eu/def#contactValue
    </externalProperty>
    <externalProperty id="b7">http://slipo.eu/def#phone http://slipo.eu/def#contactValue
    </externalProperty>
  </rule>
  <defaultDatasetAction>keep-left</defaultDatasetAction>
</rules>
```

Finally, the equivalent fusion rule for phone properties can be constructed through the web interface as presented in Figure 10.

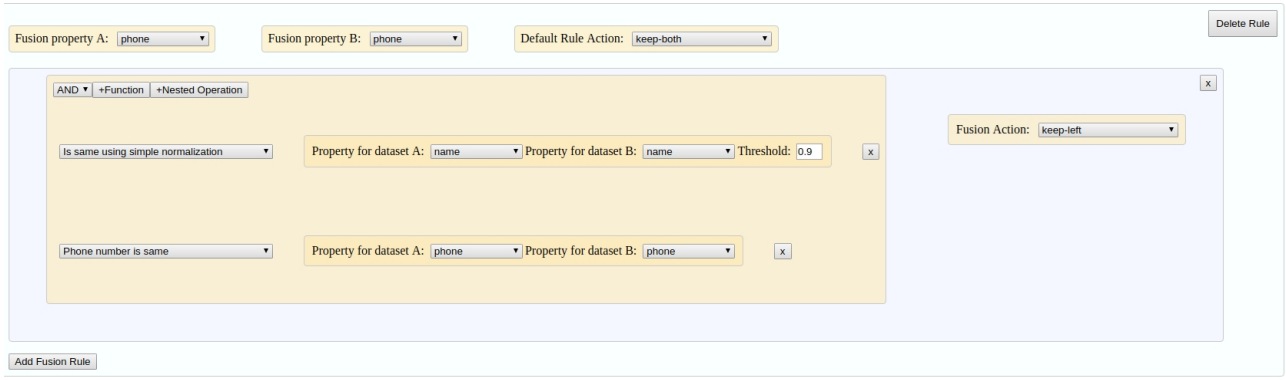


Figure 10: UI for fusion rule specification on POI phones

5.3.3. FAGI execution

Upon the configuration and rule specification files are prepared, the user can deploy FAGI. The deployment can be performed either through the SLIPO Workbench, within a designed integration workflow, or using the instructions of Section 5.1. The output is produced in the directory prescribed in the configuration file and consists of an RDF file containing the fusion results and a file containing the produced statistics (see Section 4.1.4).

6. Experimental Evaluation

In this section, we present the experiments that assess the efficiency and scalability of FAGI v2.0. In our experiments, we evaluate both the centralized version of FAGI, as well as its deployment using the distributed execution scheme (Section 4.2.3). First, we briefly present the POI datasets we used in our evaluation, that span from 100K links of POIs (~1.65M triples per dataset) to 10M links of POIs (~190M triples per dataset). Next, we compare the current version of FAGI (v2.0) with its initial version, before the start of SLIPO project (v1.0) and demonstrate the increase in efficiency in v2.0, despite the increased functionality compared to FAGI v1.0. This experiment assesses the performance of the software in a centralized execution setting, handling 100K links of POIs. Finally, we assess the scalability of FAGI v2.0, by measuring execution times when handling 1M and 10M links of POIs, using our custom distributed execution scheme. This experiment demonstrates that FAGI v2.0 can fuse, in acceptable times, millions of linked POIs.

6.1. Datasets and measures

6.1.1. Datasets

In our experiments, we use the TomTom (dataset A) and Herold (dataset B) POI datasets for Austria, that are described in the Annex (Section 8.1). These initially include 312,385 and 350,053 POIs respectively. Since in these experiments we need to evaluate efficiency and scalability, we synthetically expanded these initial datasets, creating in total three versions of each, presented in Table 11.

		Dataset A		Dataset B	
	Links	POIs	Triples	POIs	Triples
Version 1	100,000	100,000	1,507,659	100,000	1,881,912
Version 2	1,000,000	1,244,312	18,750,588	1,400,212	26,354,932
Version 3	10,000,000	10,888,737	164,082,950	12,251,855	230,605,655

Table 11: Evaluation datasets

The links between the datasets were also synthetically created using LIMES (with very loose similarity thresholds, so that many POIs end up interlinked)[14]. The three produced versions allow us to evaluate the performance of FAGI in different fusion settings (i.e., city-level, country-level, continent-level fusion respectively).

Version 1 of the datasets was used to compare the centralized version of FAGI v2.0 with FAGI v1.0. Versions 2 and 3 of the datasets were applied to evaluate the scalability of the distributed deployment of FAGI v2.0, since the centralized version of FAGI v2.0 and FAGI v1.0 was (as expected) not able to handle the respective dataset sizes.

[14] These experiments regard efficiency and scalability; hence, ensuring that the utilized links between POIs are correct/accurate was not a concern.

6.1.2. Measures

In the performed experiments, we measure execution times (*in seconds*), for several parts of the fusion process. Specifically, we report execution times for the following quantities-subtasks of the fusion process:

- **Loading time**: Measures the time required by each version of FAGI in order to load the input datasets (two input POI dataset files and the corresponding links file).
- **Fusion time**: Measures the time required for FAGI to perform the actual fusion task, including (*depending on the version of the software*), rule specification parsing, transformation of the data to the proper internal formats, and writing the produced fused RDF triples into the output file.
- **Partitioning time** (distributed FAGI v2.0 only): It reports the time required by our custom partitioning scheme to transform the initial input datasets into separate partitions to be parallely processed by multiple FAGI v2.0 instances.
- **Data transfer time** (distributed FAGI v2.0 only): It reports times measuring the data transfer overhead between nodes.
- **Total time**: The total execution time, including all applicable subtask times.

6.2. Results

The following Table 12 demonstrates the execution times of the different versions of FAGI (FAGI v1.0, FAGI v2.0-centralized, FAGI v2.0-distributed) on the three different versions of the input datasets, represented by their corresponding links (100K, 1M and 10M links respectively). We note that:

- When one of the above measures is not applicable in a specific execution combination, the measure value is marked as "-".
- When a version of FAGI did not manage to complete execution of a specific subtask, it is marked as *DNF*.

The three dataset versions are represented by the number of links they contain. The first comparison regards the scenario of POI data fusion in a city-level (100K links). In this setting, we compare the centralized current version of FAGI (v2.0) with the initial version of the software. We can observe that, although FAGI v2.0 incorporates *much richer functionality*, it achieves fusion in *less than one third of the time* required by FAGI v1.0 (132 sec vs 474.9 sec). These numbers demonstrate the effectiveness of the optimizations and code restructuring/enhancements we performed during the first period of the project. Further, it demonstrates that FAGI v2.0 requires *only a bit more than two minutes* to perform fusion on 100K linked POIs, making it rather efficient for commercial dataset sizes.

The second comparison examines the setting of 1M links (*~45M triples in both input datasets in total*). In this setting, the distributed deployment mode of FAGI v2.0 is compared to the initial version of the software (v1.0). We note that FAGI v1.0 did not support a distributed execution mode and was depended on the Virtuoso RDF store, which was used for loading and querying the input data. In this experiment, we aim to assess the gains of developing and applying a custom partitioning and distributed processing scheme, against the initial, centralized version.

Indeed, Table 12 shows that the distributed deployment of FAGI v2.0 is *more than 100x faster* than FAGI v1.0 in fusing 1M POIs (262.1 sec vs. 27378.2 sec). In the specific setting, FAGI occupies only ten nodes, each one running an individual FAGI instance. Further, in this setting FAGI v2.0 requires *less than five minutes* to perform fusion on 1M linked POIs, which corresponds to a country-level fusion process.

Finally, the third comparison examines the setting of 10M links (~394M triples in both input datasets in total). Again, the distributed deployment mode of FAGI v2.0 is compared to the initial version of the software (v1.0). As expected, FAGI v1.0 cannot complete its execution after several hours of running. On the contrary, the distributed deployment of FAGI v2.0 requires 1422 sec (24 min) to fuse 10M POIs. This is an acceptable runtime for the considered dataset sizes, taking also into account that fusion is a more fine-grained process, and thus, is usually performed individually in smaller scale datasets. However, one of our goals for the next period is to further improve the reported numbers and make FAGI a robust and efficient software, both in centralized and in distributed deployment settings.

Dataset and software versions	Partitioning time (sec)	Data transfer time (sec)	Loading time (sec)	Fusion time (sec)	Total time (sec)
100,000 links					
FAGI v1.0	-	-	19.6	455.3	474.9
FAGI v2.0 centralized	-	-	29.6	102.4	132.0
1,000,000 links					
FAGI v1.0	-	-	899.2	26479.0	27378.2
FAGI v2.0 distributed	109.0**	18.3	30.8*	104.0*	262.1
10,000,000 links					
FAGI v1.0	-	-	*DNF*	*DNF*	*DNF*
FAGI v2.0 distributed	966.2**	141.8	69.8	244.2	1422.0

Table 12: Runtimes (in sec.) of FAGI versions

**Max value from all nodes reported to represent the slowest processing node/FAGI instance, in a parallel processing setting.*

*** Partitioning times also include the runtimes for the inverse process, i.e. merging of the individual, fused files*

7. References

[BN08] Bleiholder, Jens and Naumann, Felix. Data fusion. In ACM Comput. Surv., v. 41:1, pp. 1-41, 2008.

[CCM+98] Cobb, Maria A., and Chung, Miyi J., and Miller, Vincent, and Foley, Harold III, and Petry, Frederick E., and Shaw, Kevin B. A Rule-Based Approach for the Conflation of Attributed Vector Data. GeoInformatica, v. 2:1, pp. 7–35, 1998.

[CK08] Chen Ching-Chien, Knoblock Craig A. Conflation of Geospatial Data. Encyclopedia of GIS, pp. 133-140, 2008.

[CSK+06] Chen, Ching-Chien, and Shahabi, Cyrus, and Knoblock, Craig A., and Kolahdouzan, Mohammad. Automatically and Efficiently Matching Road Networks with Spatial Attributes in Unknown Geometry Systems. In Proceedings of the Third Workshop on Spatio-temporal Database Management (co-located with VLDB2006), 2006.

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

[FAGI-api] Athena R.C. FAGI API: https://slipo-eu.github.io/FAGI/apidocs/index.html

[FAGI-doc] Athena R.C. FAGI documentation: https://slipo-eu.github.io/FAGI/dependencies.html

[FAGISPEC] Athena R.C. FAGI specification constants: https://github.com/SLIPO-EU/FAGI/blob/master/src/main/java/gr/athena/innovation/fagi/specification/SpecificationConstants.java

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

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

[GeoKnowD323] GeoKnow EU/FP7 project. D3.2.3 Fusing of geospatial relations.

[GVK+15] Giorgos Giannopoulos, Nick Vitsas, Nikos Karagiannakis, Dimitrios Skoutas, Spiros Athanasiou FAGI-gis: A Tool for Fusing Geospatial RDF Data. In The Semantic Web: ESWC 2015 Satellite Events, 2015.

[MMB12] Mendes, P. N. and Muhleisen, H. and Bizer, C. Sieve: linked data quality assessment and fusion. In Proc. of the Joint EDBT/ICDT Workshops, pp. 116-123, 2012.

[LBB17] Oliver Lehmberg, Christian Bizer, Alexander Brinkmann: WInte.r - A Web Data Integration Framework. International Semantic Web Conference (Posters, Demos & Industry Tracks), 2017.

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

[MK+12] Michelfeit, J. and Knap, T. Linked Data Fusion in ODCleanStore. In Proc. of the International Semantic Web Conference (Posters & Demos), 2012.

[MONGE] The field matching problem: Algorithms and applications. Second International Conference on Knowledge Discovery and Data Mining (KDD), 1996.

[PSC+16] Comparative Analysis of Approximate Blocking Techniques for Entity Resolution. Proceedings of the VLDB Endowment, 2017

[Saa88] Saalfeld, Alan. Conflation: Automated Map Compilation. Int. J. Geogr. Inf. Sci. v. 2:3, pp. 217–228, 1988.

[SCIK] scikit-learn - Machine Learning in Python. http://scikit-learn.org/stable/index.html

[SFM17] Learning to combine multiple string similarity metrics for effective toponym matching. International Journal of Digital Earth, 2017.

[SLIPOD11] SLIPO EU/H2020 project. D1.1 Use Cases and Requirements.

[SLIPOD12] SLIPO EU/H2020 project. D1.2 Architecture.

[SLIPOD13] SLIPO EU/H2020 project. D1.3 Beta SLIPO integrated system.

[SLIPOD22] SLIPO EU/H2020 project. D2.2 Mapping specification and POI transformation service.

[SLIPOD31] SLIPO EU/H2020 project. D3.1 Interlinking.

[SMI+12] Schultz, A. and Matteini, A. and Isele, R. and Mendes, P. and Bizer, C. and Becker, C. LDIFA Framework for Large-Scale Linked Data Integration. In WWW, developer track, 2012.

[WF99] Walter, Volker, and Fritsch, Dieter. Matching Spatial Data Sets: a Statistical Approach. Int. J. Geogr. Inf. Sci. v. 5:1, pp. 445--473, 1999.

8. Annex

8.1. Datasets characteristics

8.1.1. Dataset A

Dataset A includes 312,385 TomTom POIs for Austria. The dataset is initially provided as a Shapefile in the WGS84 reference system. The schema of the dataset, i.e. the properties of the contained POIs are provided in the table below.

Property Name	Description
OBJECTID	POI ID
NAME	POI name
A_NAME	Alternative POI name
STR	Street name
HNR	Street number
ZIP	ZIP code
A00	Country code
A08_NAME	city/place
LABEL_DE	Description of POI (ie class)
LABEL_EN	Description of POI (ie class)
PHONE	Telephone
BRAND	Brand name for chains
ACCURACY	Geographic accuracy
SUBCAT	Category by TomTom
SUBNAME_DE	Classification by TomTom
SUBNAME_EN	Classification by TomTom

8.1.2. Dataset B

Dataset B contains all company POIs available in the Herold Business Data for Austria database, summing to 350,053 records. The dataset is initially provided as a Shapefile in the WGS84 reference system. The schema of the dataset, i.e. the properties of the contained POIs are provided in the table below.

Property Name	Description
SID	Company ID
FIRMA	Company name
BR	Branch of business

PLZ	ZIP code
ORT	City/place
STRASSE	Street name
HNR	House number
BL	Country
TELEFON	Phone
TELEFAX	Fax
HTTP	Homepage
EMAIL	Email
OENACE	Categorization
OENACE_ERW	Categorization
OENACE_BEZ	Categorization
BR_ID	HEROLD's branch of business ID
HIC	Branch group