DELIVERABLE D3.4

Initial Release of the Evolution 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.

Dissemination Level	Public
Due Date of Deliverable	Month 24 (31/12/2018)
Actual Submission Date	21/12/2018
Work Package	WP3 – Data Integration and Quality Assurance
Tasks	Task 3.4: Provenance and Co-Evolution
Type	Other
Approval Status	Submitted for approval
Version	1.0
Number of Pages	20
Filename	D3.4_Initial_Release_of_the_Evolution_Framework.pdf

Abstract

This report presents the status of our work on monitoring POI data evolution. In particular, we have implemented a service that allows to track the integration and evolution of POI information across time and between different versions. This includes mechanisms for recording the provenance of the data, so that the full history of changes in the available information about a POI can be tracked, starting from the first version of the input about a particular POI, up to the current values of its various properties. We also present a graphical user interface for visualizing and navigating through this information, so that the user can explore this information easily and intuitively without being exposed to technical details.

History

version	date	reason	revised by
0.1	12/11/2018	First draft	Dimitris Skoutas
0.2	19/11/2018	Input to Sections 3 and 4	Yannis Kouvaras, Nikos Karagiannakis, Dimitris Skoutas
0.3	1/12/2018	Contributions to all sections	Damien Graux, Pauline Baudens
0.4	4/12/2018	Updated screenshots, multiple edits	Yannis Kouvaras, Nikos Karagiannakis, Dimitris Skoutas
0.5	18/12/2018	Updated introduction and figures	Dimitris Skoutas
0.9	20/12/2018	Minor edits in Section 4	Damien Graux, Pauline Baudens
1.0	21/12/2018	Completed version for submission	Spiros Athanasiou

Author list

organization	name	contact information
ATHENA RC	Spiros Athanasiou	spathan@imis.athena-innovation.gr
ATHENA RC	Yannis Kouvaras	jkouvar@imis.athena-innovation.gr
ATHENA RC	Nikos Karagiannakis	nkaragiannakis@imis.athena-innovation.gr
ATHENA RC	Dimitris Skoutas	dskoutas@imis.athena-innovation.gr
IAIS	Damien Graux	damien.graux@iais.fraunhofer.de
TomTom	Pauline Baudens	pauline.baudens@tomtom.com

Executive Summary

This report presents the current status of the Evolution Framework implemented in SLIPO for tracking the evolution and provenance of POI information throughout POI data integration workflows designed and executed in the SLIPO Workbench. The evolution framework tracks, manages, and presents the provenance of POIs following a data integration workflow at the POI and attribute level, enabling end-user to know exactly from *where* and *why* a specific piece of information was added in the final output dataset (*batch updates*). Due to the sophisticated management and versioning capabilities of our workflow engine, we are also able to keep track of evolution of POI entities for subsequent executions of the same workflow (delta updates), thus enabling the end user to understand the impact of a multiple executions (*e.g., with different assets, pre-processing, configurations*) at the POI level. Provenance information is conveyed to users following the established paradigm of interactive maps and attribute tables enabling free roaming (*zoom, pan, drill-in*) across the data integration space, thus allowing them to view a specific POI and assess the operation parameters (*configurations*) of a particular integration workflow. This essentially enables them to visually query the underlying assets regarding the origin of their attributes and their changes and track their evolution over time

With these foundations in place, our ongoing work focuses on providing users with advanced querying and editing capabilities, in order to offer a complete and streamlined environment for tracking and editing the output of the data integration process. Specifically, we are developing functionality for performing queries and providing statistics regarding the evolution of POI metadata and their arithmetic properties. Further, we will enable users to *edit* a specific POI attribute, either to improve the output itself (e.g., correct outliers) or to inform subsequent executions of the data integration workflow (i.e., training data for automated mode). This line of work aims to closely resemble the standard quality assurance processes typically employed pre and post data integration (*i.e., view in a GIS, manually edit table*) and provide a holistic user experience. Combined with the quality assurance metrics presented in Deliverable 3.4, our ambition is to ultimately negate the need for the user visiting external software or employing processes outside the SLIPO system to view, asses and improve the data integration output. This not only makes our software more useful and relevant on a real-world setting, but also allows us to capture and leverage the manual edits to improve the automatic operation of the various SLIPO toolkit components.

The current version of the evolution framework has already been deployed and is being actively used in the context of our pilot, with the users comparing the SLIPO service to their standard operating processes, further feeding and guiding our work.

The deliverable is structured as follows:

In Section 1 we introduce the main concepts of this work, outlining its scope and goals.

In Section **Error! Reference source not found.** we explain where and how changes in the values of POI attributes occur throughout POI data integration workflows in SLIPO and present the additional functionality implemented in FAGI for keeping track of provenance information when executing fusion actions.

In Section 3 we describe the extensions that have been implemented in the SLIPO Workbench to allow for storing and querying this provenance information, as well as for visually presenting it to the users.

Abbreviations and Acronyms

CSV	Comma-Separated Values
NFS	Network File System
POI	Point of Interest
RDF	Resource Description Framework
UI	User Interface
XML	eXtensible Markup Language

Table of Contents

1. Introduction

POI entities are by nature *temporally evolving*. The profile of a POI may often contain information about a large number of properties characterizing that POI. Many of these properties are dynamic, having values that vary or change frequently; for instance, the availability of a hotel or a parking space, the items in a restaurant's menu, the special offers provided by a shop, etc. Even other attributes that are considered to be rather static, such as the name of a POI, its category, its address, etc., may still change over time. Thus, the information stored about a POI reflects the status of that POI at some given point in time. In other words, we can think of it as a *snapshot* at some certain point in time during that POI's life. If multiple such snapshots about the same POI exist, i.e., data about this POI has been collected at various points in time, either periodically or on demand, the need arises to be able to *track the changes* of the values of various attributes of the POI throughout time.

Moreover, changes to the data about a POI may occur even if the POI itself has not changed. In that case, these changes reflect how *our own knowledge* about this POI has changed over time, even if the actual characteristics of the POI may have remained unchanged. In fact, this is inherent to the processes of collecting data about POIs. Depending on the type of this process, and various related factors, the degree of completeness and uncertainty of the values of the various attributes varies. For instance, data that has been collected via field work is subject to *missing values and inaccuracies* introduced by human errors; data collected from the Web or other sources is subject to errors introduced by the information retrieval and extraction process, as well as any errors existing in those sources already.

By applying the SLIPO workbench to perform a series of POI data integration workflows, producing new integrated POI datasets out of two or more initial ones, users can increase POI data quality by updating the values of POI attributes to more recent snapshots and/or filling in missing values or correcting erroneous ones. While doing so, it is desirable, and often necessary, to keep track of changes in the values of POI attributes. In this way, the user can determine the *provenance* of each piece of information, which in other words means to be able to tell, at any given snapshot and for any given POI attribute, when and how that particular value came to be.

In this report, we present the mechanism we have implemented so far for assisting the users of the SLIPO workbench to monitor and track the evolution and provenance of POI information throughout the performed data integration cycles. We explain what information is maintained, for which parts of the data integration workflow, how it is stored and queried, and how it is presented to the user. As with other components of the SLIPO Workbench, the emphasis lies again on hiding the complexities and technicalities of the underlying mechanism from the user, allowing her to navigate through this information in an intuitive and visual manner.

2. Tracking changes in SLIPO Workflows

2.1. Occurrence of changes

The SLIPO Workbench allows to design and execute POI data integration workflows that are composed by one or more of the following operations:

- *transformation*, executed by TripleGeo;
- *interlinking*, executed by LIMES;
- *fusion*, executed by FAGI; and
- *enrichment*, executed by DEER.

Inspecting each of these operations, it is easy to observe that the first two, namely *transformation* and *interlinking*, do not introduce changes in the values of POI attributes. Specifically, TripleGeo performs syntactic transformation of POI representations from their original schema and format (e.g., a CSV, XML or JSON file, or a spatial database) into RDF triples. During this process, the representation of the POI data changes but the values of the properties remain the same. Similarly, LIMES receives as input two POI datasets A and B, and generates *sameAs* links between pairs of POIs from A and B. Again, the original values of any two interlinked POIs remain the same.

However, changes occur during the subsequent two operations, namely *fusion* and *interlinking*. In particular, FAGI receives as input two interlinked POI datasets A and B, and produces as output a third POI dataset C that is the result of merging A and B together based on the identified links and a specified set of fusion rules. During this process, the resulting value of a property of a given POI in dataset B may be either one of the initial values of that POI in dataset A or B or potentially a new value produced by a function that combines the two original ones. Similarly, DEER receives as input an initial POI dataset and produces as output an enriched one where the input POIs have been enriched by adding missing values or replacing existing ones based on information retrieved from external sources.

From the above, it becomes clear that in order to track the evolution and provenance of POI data throughout executions of POI data integration workflows in SLIPO, it is necessary and sufficient to record and monitor changes performed by FAGI and DEER. In this report, we focus on the former. We explain the work that has been done so far to enable the tracking of provenance information in FAGI, as well as the extensions implemented in the SLIPO Workbench to enable storing, querying and visualizing this information.

2.2. Tracking Changes in FAGI

As mentioned above, FAGI receives as input two POI datasets A and B, as well as a set L of *sameAs* links between them (produced by LIMES). Then, FAGI merges the two datasets into a fused dataset C. The actions performed during fusion are driven by a set of specified fusion rules. These rules may have been crafted manually or may have been automatically learned from previous executions. Each rule comprises zero or

more conditions (evaluated over the values of POI attributes in A and/or B), accompanied by corresponding fusion actions. These functions may indicate to maintain either of the initial values in A or B, or to somehow combine both values into constructing a new one (e.g., by string concatenation, or numeric aggregation).

To allow for tracking changes and their provenance, FAGI has been extended with a mechanism that records all such relevant value transformations. This is then provided as additional output, alongside the merged dataset C. Specifically, for each pair of interlinked POIs P_i in A and P_j in B, this additional output comprises statements including the following information:

- the identifiers of P_i and P_j;
- the name of the property on which the fusion action was applied;
- the fusion action that was applied;
- the original values of this property in each POI;
- the fused value that was produced as result;
- a confidence score for the action performed (if available).

More specifically, FAGI implements a logging mechanism that keeps track of the fusion actions it performs. This mechanism is optional and can be turned on or off by the user by setting the value of the parameter *verbose* to true or false, respectively, in FAGI's configuration file. This fusion log keeps track of the fusion actions applied on *each* attribute of *each* pair of linked POIs, as well as a fusion confidence score for each pair. The fusion log is written as a separate output file.

As an example, consider a pair of interlinked POIs A and B with a set of attributes (represented as key-value pairs) as shown below.

POI A	POI B
uri: A *name*: "81ο Δημοτικό Σχολείο Αθηνών" *phone*: 2103466031 *address-street*: "Σκαμβωνιδών"@el *address-number*: 46 *postcode*: 11853 *lat*: 37.97 *lon*: 23.70 *geometry*: <http://www.opengis.net/def/crs/EPSG/0/4326> POINT(23.70 37.97)"^^<http://www.opengis.net/ont/geosparql#wktLiteral> .	*uri*: B *name_en*: "81o Primary School"@en *phone*: 2103466031-5 *address-street*: "Skamvwnidwn"@en *geometry*: <http://www.opengis.net/def/crs/EPSG/0/4326> POLYGON ((23.7069502 37.971373400000004, 23.707348600000003 37.9712224,...))"^^<http://www.opengis.net/ont/geosparql#wktLiteral> . *country*: Greece

Suppose that the user wants to merge the information about these two POIs according to the following guidelines for each attribute:

- *geometries*: keep the most detailed geometry;
- *names*: keep the longest one;

- *phones*: if both POIs have phone number, and these numbers are the same under some specified "phone-normalization" function, then the first value (i.e., from POI A) should be kept. Otherwise, both numbers should be kept.
- *address-street*: If the first POI has an address-street value, keep this value, otherwise keep the value of the second POI.
- *address-number*: First check the address-street attribute in order to avoid mixing different streets with different numbers. If the first address-street value was kept, check if there is a value of the address-number in the first POI and keep that. Otherwise, the address-number of the first POI is missing, hence the value of POI B should be kept.

Below is the formal specification of the corresponding fusion rules provided to FAGI as input configuration in XML format.

Example specification of fusion rules for merging POIs A and B.

```
<?xml version="1.0" encoding="UTF-8"?>
<rules>
 <rule>
  <propertyA>http://www.opengis.net/ont/geosparql#hasGeometry
http://www.opengis.net/ont/geosparql#asWKT</propertyA>
  <propertyB>http://www.opengis.net/ont/geosparql#hasGeometry
http://www.opengis.net/ont/geosparql#asWKT</propertyB>
  <defaultAction>keep-more-points</defaultAction>
 </rule>
 <rule>
  <propertyA>http://slipo.eu/def#name</propertyA>
  <propertyB>http://slipo.eu/def#name</propertyB>
  <defaultAction>keep-longest</defaultAction>
 </rule>
 <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>concatenate</defaultAction>
  <actionRuleSet>
        <actionRule>
                <condition>
                        <expression>
                                <and>
                                        <function>exists(a)</function>
                                        <function>exists(b)</function>
                                        <function>isSamePhoneNumberCustomNormalize(a,
b)</function>
                                </and>
                        </expression>
                </condition>
                <action>keep-left</action>
        </actionRule>
  </actionRuleSet>
 </rule>
 <rule>
  <propertyA>http://slipo.eu/def#address http://slipo.eu/def#street</propertyA>
  <propertyB>http://slipo.eu/def#address http://slipo.eu/def#street</propertyB>
        <actionRuleSet>
                <actionRule>
```

```
                        <condition>
                                <function>exists(a)</function>
                        </condition>
                        <action>keep-left</action>
                </actionRule>
        </actionRuleSet>
  <defaultAction>keep-right</defaultAction>
 </rule>
 <rule>
  <propertyA>http://slipo.eu/def#address http://slipo.eu/def#number</propertyA>
  <propertyB>http://slipo.eu/def#address http://slipo.eu/def#number</propertyB>
        <actionRuleSet>
                <actionRule>
                        <condition>
                                <function>exists(a1)</function>
                        </condition>
                        <action>keep-left</action>
                </actionRule>
                <actionRule>
                        <condition>
                                <function>exists(a)</function>
                        </condition>
                        <action>keep-left</action>
                </actionRule>
        </actionRuleSet>
  <externalProperty id="a1">http://slipo.eu/def#address http://slipo.eu/def#street</externalProperty>
  <externalProperty id="b1">http://slipo.eu/def#address http://slipo.eu/def#street</externalProperty>
  <defaultAction>keep-right</defaultAction>
 </rule>
 <defaultDatasetAction>keep-both</defaultDatasetAction>
</rules>
```

Given the two POIs A and B, and the above fusion rules, the resulting fused POI will look like this:

Fused information from POIs A and B.
name: "81ο Δημοτικό Σχολείο Αθηνών" *phone*: 2103466031, 2103466031-5 *address-street*: "Σκαμβωνιδών"@el *address-number*: 46 *postcode*: 11853 *country*: Greece *lat*: 37.97 *lon*: 23.70 *geometry*: <http://www.opengis.net/def/crs/EPSG/0/4326> POLYGON ((23.7069502 37.971373400000004, ...))"^^<http://www.opengis.net/ont/geosparql#wktLiteral> .

Moreover, FAGI will produce the following fusion log.

Fusion log for merging POIs A and B.
FusionInfo{leftURI=A, rightURI=B, actions=[Action{ attribute=http://www.opengis.net/ont/geosparql#hasGeometry http://www.opengis.net/ont/geosparql#asWKT, fusionAction=keep-more-points,

```
                    valueA=<http://www.opengis.net/def/crs/EPSG/0/4326> POINT(23.70
37.97)"^^<http://www.opengis.net/ont/geosparql#wktLiteral>,
                    valueB=<http://www.opengis.net/def/crs/EPSG/0/4326> POINT (6.764635000000055
49.836753000000044)^^http://www.opengis.net/ont/geosparql#wktLiteral,
                    fusedValue=geometry: <http://www.opengis.net/def/crs/EPSG/0/4326> POLYGON
((23.7069502 37.971373400000004, ...))"^^<http://www.opengis.net/ont/geosparql#wktLiteral> .
          },
          Action{
                    attribute=http://slipo.eu/def#name http://slipo.eu/def#nameValue,
                    fusionAction=keep-longest,
                    valueA="81ο Δημοτικό Σχολείο Αθηνών",
                    valueB="81o Primary School"@en,
                    fusedValue="81ο Δημοτικό Σχολείο Αθηνών"
          },
          Action{
                    attribute=http://slipo.eu/def#source http://slipo.eu/def#poiRef,
                    fusionAction=keep-left,
                    valueA=A,
                    valueB=B,
                    fusedValue=A
          },
          Action{
                    attribute=http://slipo.eu/def#address http://slipo.eu/def#street,
                    fusionAction=keep-left,
                    valueA="Σκαμβωνιδών"@el,
                    valueB="Skamvwnidwn"@en,
                    fusedValue="Σκαμβωνιδών"@el
          },
          Action{
                    attribute=http://slipo.eu/def#address http://slipo.eu/def#number,
                    fusionAction=keep-left,
                    valueA=46,
                    valueB=null,
                    fusedValue=46
          },
          Action{
                    attribute=http://slipo.eu/def#phone http://slipo.eu/def#contactValue,
                    fusionAction=keep-longest,
                    valueA=2103466031,
                    valueB=2103466031-5,
                    fusedValue=2103466031, 2103466031-5
          },
],
defaultFusionAction=keep-both,
validationAction=accept
fusion-confidence=0.89
}
```


This fusion log contains detailed information about all the actions that were applied based on the rule specifications. Each rule defined for a specific attribute has its own record containing the initial values, the fusion action that was applied, and the final, fused value. The attributes without rule definitions also appear at the fused POI record, as they follow the default dataset action, which in this case is to keep both values from POIs A and B.

3. Monitoring Evolution and Provenance in the SLIPO Workbench

As already explained, an important goal in SLIPO is to hide the underlying complexity and technicalities of the POI data integration process, making the results easily accessible to the user; the same holds for the underlying mechanisms tracking evolution and provenance.

To this end, we have extended the SLIPO Workbench with additional functionality and visualizations that allow the user to inspect the changes that have occurred to the values of any attribute of any POI throughout its lifetime within the SLIPO system.

In the next sub-sections, we present the implementation details on how SLIPO platform handles POI provenance and evolution data. First, we discuss how the specific data is extracted, stored and queried from existing POI data integration workflows. Next, we enumerate several UI components added to the SLIPO workbench web application for visualizing and querying this data.

3.1. Data Management

In SLIPO, every POI data integration workflow consists of several resources and steps. A resource is any RDF dataset processed during the execution of a workflow. The supported resources are:

- external files with a format supported by the TripleGeo SLIPO Toolkit component;
- any intermediate step output; or
- the output of previously executed workflows.

All resources are stored in a distributed file system (NFS) and are accessible by all steps of a workflow execution. A step represents the execution of a single SLIPO Toolkit component. During the execution of a step, the SLIPO service maintains additional runtime data that is persisted in a relational database. For each step the following data is stored:

- the start and completion timestamp;
- the step execution status;
- a set of all files generated during the step execution either by the SLIPO Toolkit component itself, e.g. statistics and output data files, or by the SLIPO service, e.g. configuration files; and
- a detailed description of any error that may has caused the step execution to fail.

The Workbench application provides the UI components required for querying and displaying step execution runtime data. Moreover, any resource or intermediate result data file can be downloaded from

the application. Additional details about workflow execution and Workbench UI can be found in deliverables D1.2 (SLIPO Architecture) and D1.3 (Beta Integrated System), respectively. Next, we focus on how provenance and evolution data is extracted from existing workflow executions.

After a POI data integration workflow is executed, all data is stored in files. In order to query and visualize provenance data efficiently, an indexing scheme must be used. To this end, we have implemented the following software components:

- **RDF Import Service**: a service for dynamically building workflows that import RDF datasets into a PostgreSQL database instance with PostGIS support; and
- **Provenance Service**: a service for querying provenance and evolution data.

The integration of the two services into the SLIPO Platform architecture is depicted in Figure 1. The RDF Import Service is part of the SLIPO Service implementation while the Provenance service is part of the SLIPO Web Application Server. In the next two sections, we describe each of these components.

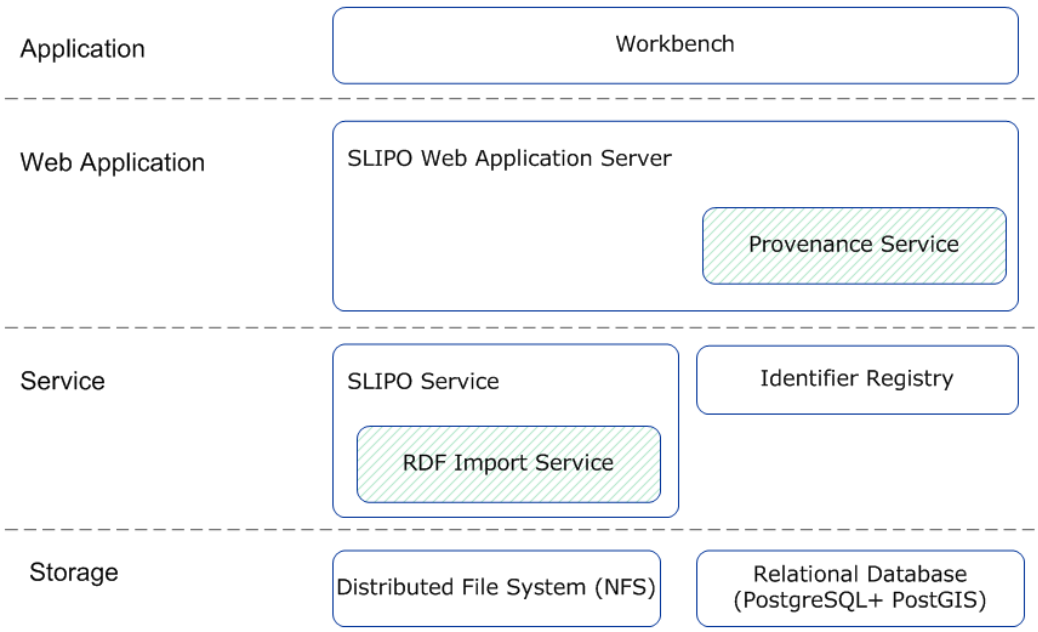


Figure 1: RDF Import Service and Provenance Service inside the SLIPO Platform

3.1.1. RDF Import Service

The RDF Import Service provides functionality for dynamically building workflows that import RDF datasets related to a single POI data integration workflow execution instance into a relation database. The generated workflows are built with the same framework used for implementing any workflow inside the SLIPO Platform and executed by the SLIPO Service. The service implements a single operation that accepts as input the unique identifier of a single workflow execution instance and creates a workflow that performs the following six tasks:

- The workflow definition and execution runtime data are retrieved and examined in order to detect all input, output and registry files. All resources that are external to the workflow, e.g. RDF datasets generated by TripleGeo transformations or results from other workflows, are categorized as input. Any RDF datasets generated by a step that are not used as input for another step are identified as output. Finally, all registry files created by TripleGeo transformations are collected.
- The input and output RDF datasets are transformed to CSV files using the TripleGeo reverse transformation functionality.
- The CSV files from the previous step are imported into a relational database. A new table is created for each file.
- The registry files are uploaded to the Identifier Registry Service.
- The workflow definition is searched for fusion operations. For each fusion operation found, the fusion log is parsed and imported into a relational database. Fusion log data is normalized into two tables, one for links and one for actions.
- The workflow execution runtime data is updated with the table names created during the previous steps.

The generated workflow currently ignores enrichment operations. Although enrichment operations may alter POI data, the service makes the following two assumptions:

- for every output file, there is no or at most one enrichment operation; and
- if an enrichment operation exists, it is the last one executed.

Hence, enrichment modifications can be deduced by inspecting input, output and fusion log files. For a workflow with M inputs, N outputs and K fusion operations, a total of $M + N + 2 * K$ tables are created.

The service can be invoked either immediately after the execution of a POI data integration workflow or on demand. Once the workflow is built and executed, the Provenance Service can be used for querying provenance and evolution data for the related workflow execution instance.

3.1.2. Provenance Service

A central feature of the SLIPO Platform is the support of workflow versioning. Every time a workflow is updated, its version is incremented, and a new workflow instance is created. A workflow instance may be executed only once, resulting into a single workflow execution instance or data integration cycle. The Provenance Service allows users to query POI updates either inside a single execution or across multiple executions of workflow versions.

Specifically, once a user selects an output POI from a specific workflow execution, Provenance Service provides the following two operations:

- query for updates applied during the specific workflow execution; and
- compare attribute values against the corresponding ones of the same POI in another workflow version execution instance.

In order to answer the former query, the Provenance Service queries the tables created by the RDF Import Service. First the output tables are queried and once the POI is found, fusion log tables are searched recursively until all input POIs are returned. For the latter case, the service first queries the Identifier Registry in order to find all possible identifiers for the selected POI. Then the output tables of the other workflow execution instance are queried using all the identifiers returned by the registry.

The UI components used for executing Provenance Service operations and displaying the results is described in the next section.

3.2. Workbench User Interface

In this section we present the Workbench User Interface (UI) for querying the Provenance Service and rendering query results. Workbench provides the following features for exploring POI provenance and evolution data:

- Building thematic maps where each input and output file is represented as a separate layer.
- Selecting one or more POIs from the map.
- Displaying provenance data for a single POI.
- Displaying evolution data for a single POI across multiple workflow version executions.

Consider the POI data integration workflow illustrated in Figure 2. This workflow receives two POI datasets as input, which are interlinked and fused. Finally, an enrichment operation is executed on the fused result.

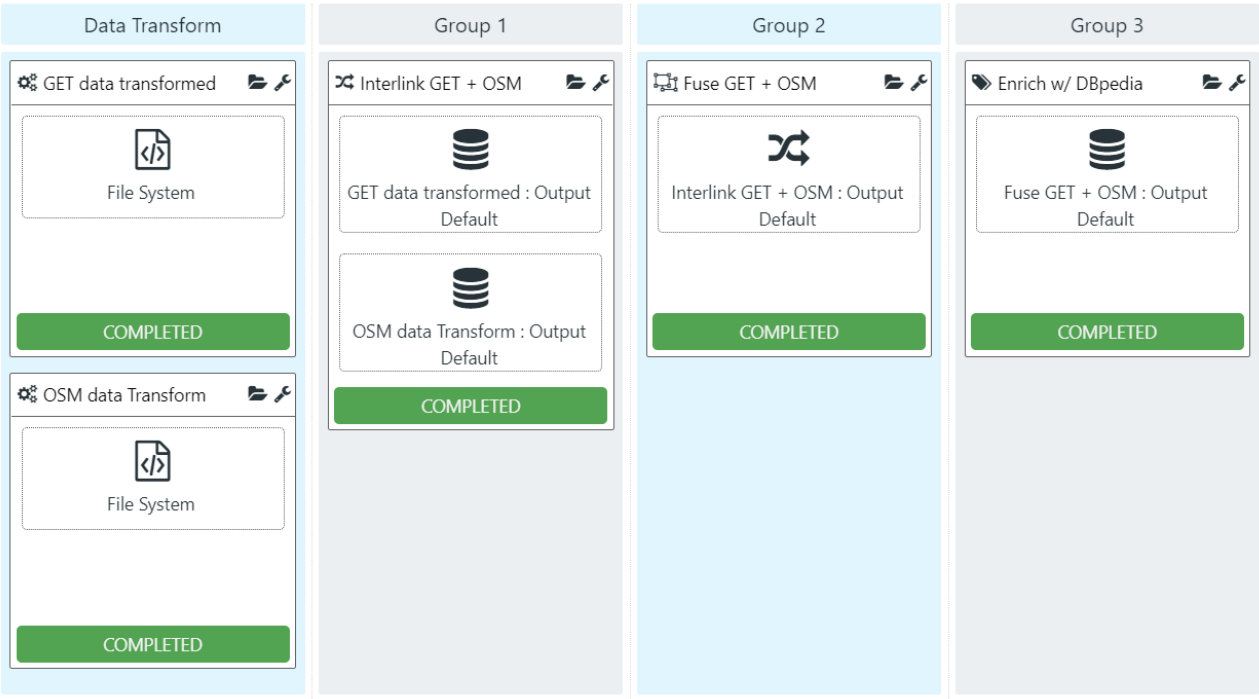


Figure 2: A POI data integration workflow.

The user can visualize the involved POIs on a map, with different layers corresponding to different input and output datasets as shown in Figure 3.

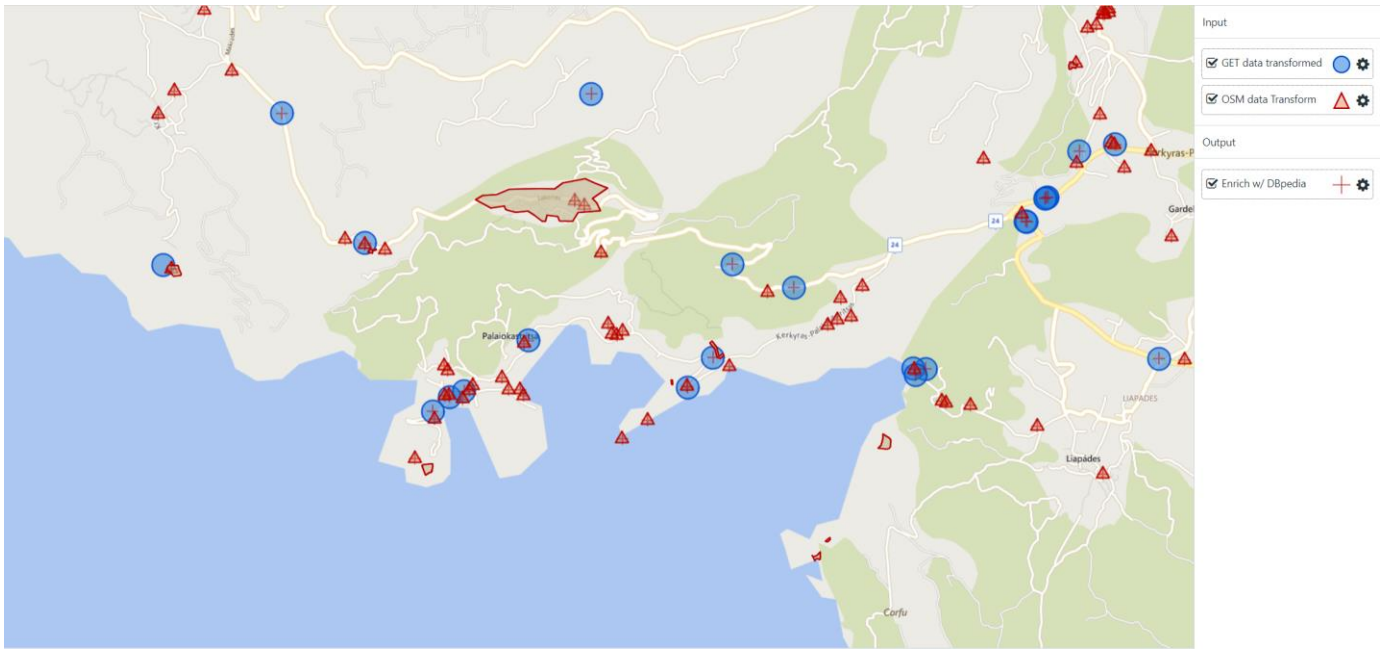


Figure 3: Map visualization of the workflow's input and output datasets.

Moreover, users can configure the appearance and style of each layer by setting various properties, such as symbols for point data or fill color for polygons, as displayed in Figure 4. Furthermore, the style of each layer is persisted with the rest of the workflow execution runtime data.

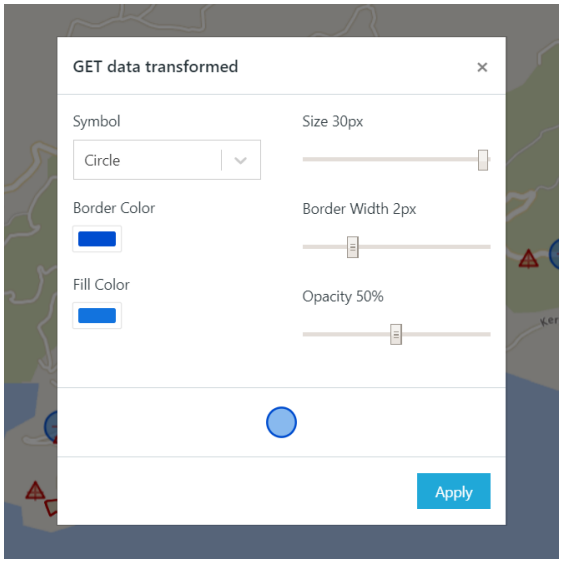


Figure 4: Layer style configuration.

By clicking on the map, the user can select a desired POI. In particular, given that often there may exist a large number of POIs located very close to each other (or even with the exact same coordinates), a popup window is displayed that allows the user to select among those POIs located close to the point that was

clicked. This is shown in Figure 5. The popup window contains a table where each record corresponds to a POI, including the respective layer that it comes from, and basic POI attributes (id, name, category, etc.).

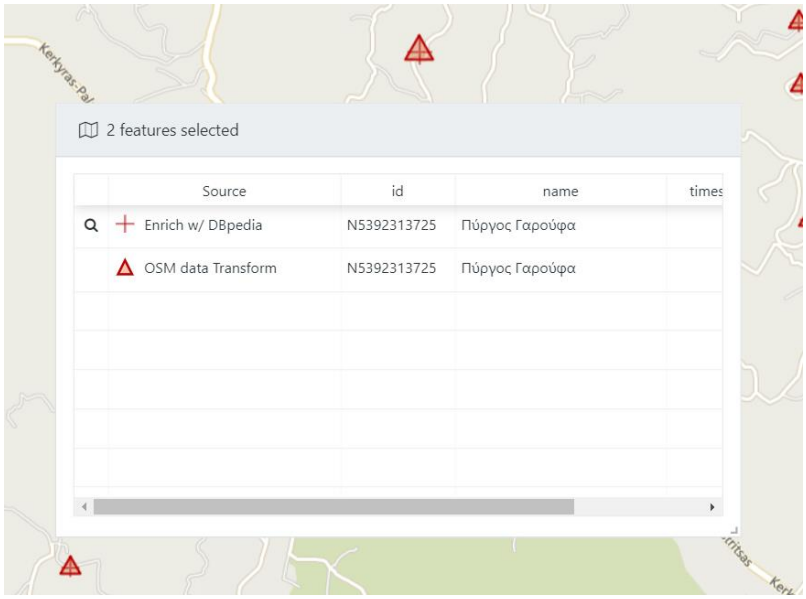
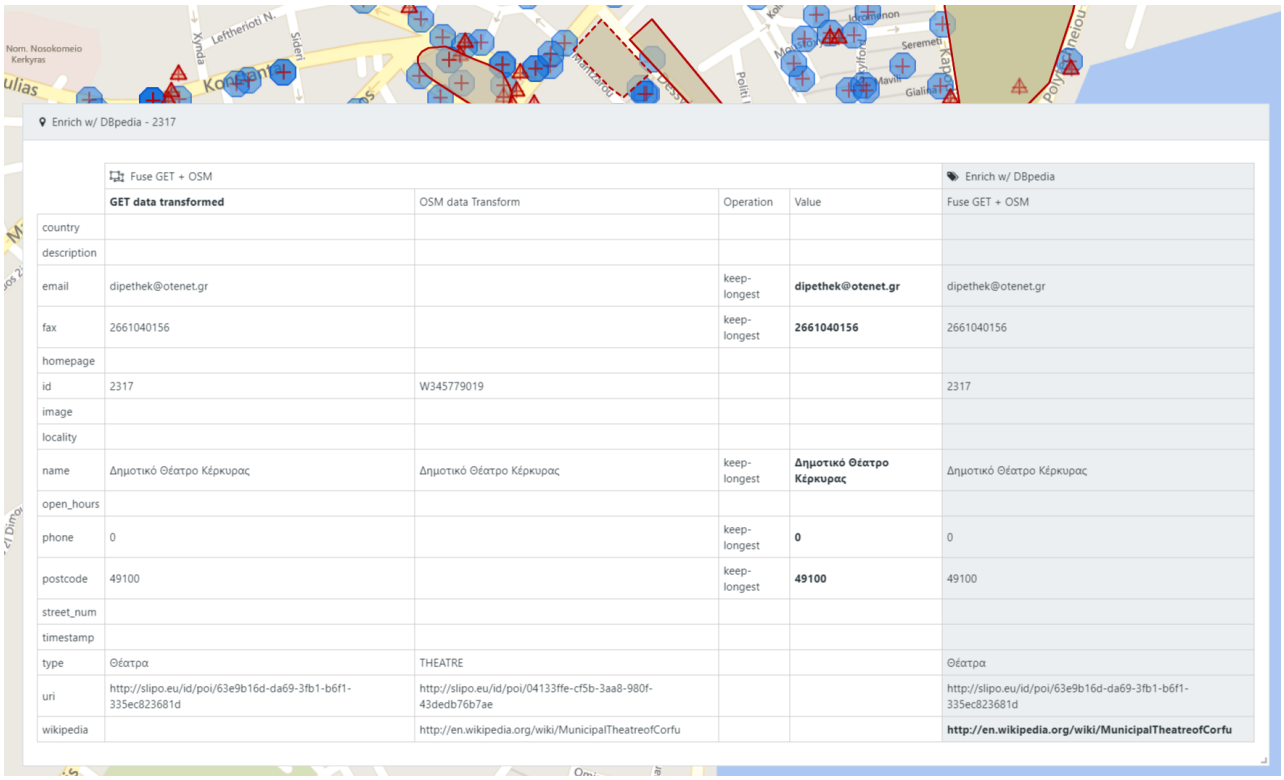


Figure 5: Selecting a POI on the map.

When the user selects an output POI, a new popup window is displayed containing the complete provenance information for this POI. This is shown in the following figure.



	Fuse GET + OSM				Enrich w/ DBpedia
	GET data transformed	OSM data Transform	Operation	Value	Fuse GET + OSM
country					
description					
email	dipethek@otenet.gr		keep-longest	**dipethek@otenet.gr**	dipethek@otenet.gr
fax	2661040156		keep-longest	**2661040156**	2661040156
homepage					
id	2317	W345779019			2317
image					
locality					
name	Δημοτικό Θέατρο Κέρκυρας	Δημοτικό Θέατρο Κέρκυρας	keep-longest	**Δημοτικό Θέατρο Κέρκυρας**	Δημοτικό Θέατρο Κέρκυρας
open_hours					
phone	0		keep-longest	**0**	0
postcode	49100		keep-longest	**49100**	49100
street_num					
timestamp					
type	Θέατρα	THEATRE			Θέατρα
uri	http://slipo.eu/id/poi/63e9b16d-da69-3fb1-b6f1-335ec823681d	http://slipo.eu/id/poi/04133ffe-cf5b-3aa8-980f-43dedb76b7ae			http://slipo.eu/id/poi/63e9b16d-da69-3fb1-b6f1-335ec823681d
wikipedia		http://en.wikipedia.org/wiki/MunicipalTheatreofCorfu			**http://en.wikipedia.org/wiki/MunicipalTheatreofCorfu**

Figure 6: Provenance information for a selected POI.

As can be seen, this table provides detailed provenance information for each individual attribute of the POI. Specifically, each record in the table corresponds to an individual attribute (e.g., name, category, etc.). The columns of the table correspond to the individual operations taking place during this workflow. By following the values in each row from left to right, we can easily track and inspect the value of the corresponding property throughout the workflow execution. Thus, we can see what the original value was in the input, and if, when, and how this value changed throughout the integration process.

In the example above, the UI highlights the following updates:

- During the fuse operation for GET and OSM datasets, the POI from the GET dataset was selected.
- Fusion actions have been applied to fields email, fax, name, phone, name and postcode. The fused values and actions are also displayed.
- In the final result, the enrichment operation has updated the missing value for the wikipedia attribute of the GET POI.

Finally, if the workflow of the selected execution has multiple versions and the corresponding executions have also been processed by the RDF Import Service, the user can traverse the maps associated to each version using the UI component shown in the bottom of Figure 7.

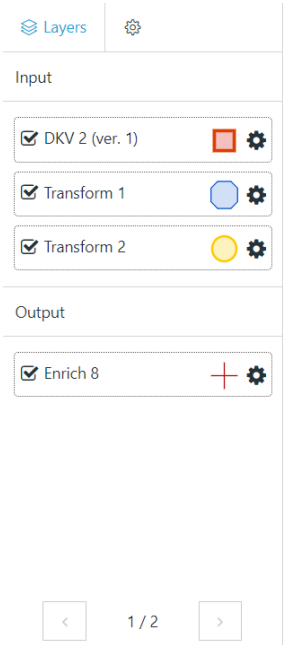


Figure 7: Navigation across different execution instances of a workflow