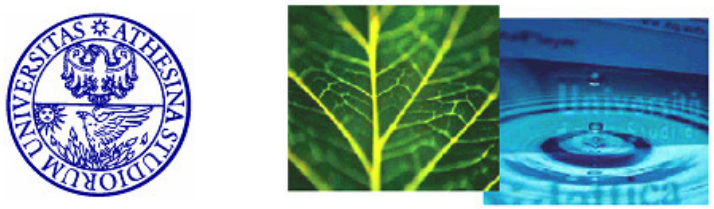
PhD Dissertation



International Doctorate School in Information and Communication Technologies

DIT - University of Trento

Provenance in Open Data Entity-Centric Aggregation

Moaz Mohammed Reyad Abdelhamid Abdelnaby

Advisor:

Prof. Fausto Giunchiglia

Università degli Studi di Trento

April 2015

Abstract

An increasing number of web services these days require combining data from several data providers into an aggregated database. Usually this aggregation is based on the linked data approach. On the other hand, the entity-centric model is a promising data model that outperforms the linked data approach because it solves the lack of explicit semantics and the semantic heterogeneity problems. However, current open data which is available on the web as raw datasets can not be used in the entity-centric model before processing them with an import process to extract the data elements and insert them correctly in the aggregated entity-centric database. It is essential to certify the quality of these imported data elements, especially the background knowledge part which acts as input to semantic computations, because the quality of this part affects directly the quality of the web services which are built on top of it. Furthermore, the aggregation of entities and their attribute values from different sources raises three problems: the need to trace the source of each element, the need to trace the links between entities which can be considered equivalent and the need to handle possible conflicts between different values when they are imported from various data sources. In this thesis, we introduce a new model to certify the quality of a back ground knowledge base which separates linguistic and language independent elements. We also present a pipeline to import entities from open data repositories to add the missing implicit semantics and to eliminate the semantic heterogeneity. Finally, we show how to trace

the source of attribute values coming from different data providers; how to choose a strategy for handling possible conflicts between these values; and how to keep the links between identical entities which represent the same real world entity.

Keywords

provenance, entity, data aggregation, knowledge base

As for the scum it vanishes, being cast off;

but as for that which benefits people, it remains on the earth.

-- Holy Quran

Acknowledgements

I would like to express my sincere gratitude to my advisor Prof. Fausto Giunchiglia for his valuable guidance, support and constructive comments throughout the journey toward my PhD.

I would also like to thank all the members of the Knowdive group at the University of Trento. I can not list all the names of the current and previous researchers, software developers and administrators of the group who contributed to my work and knowledge.

Finally, I can not forget to thank my thesis committee for their time and effort.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 The Context

Recently an increasing number of open data catalogs appear on the Web [54]. These catalogs contain data that represents real world entities and their attributes. Data can be imported from several catalogs to build web services. The acceptance of these web services has several quality requirements. One of these requirements is the certification of data quality. Data can be seen as the food of the web. Restaurants that offer high quality food may offer to the customers some facts about the source of their ingredients as a sign of quality and to gain their trust. For the web, data is the food which is continuously produced and consumed by web services and users. Effective tracing for the sources of data on the web plays a major role in the acceptance of the services built on top of it.

Web services in future will be based on linked data and linked entities. We refer to real world objects that are of enough importance to be given a name as *entities*. Examples for entities are **Italy** and **Barack Obama**. There are different types of entities, such as Locations and Persons. Italy is an entity of type **Location** and Barack Obama is an entity of type **Person**. The aggregation of data in our approach is entity-centric, i.e. it imports data from external sources as entities and entity attributes. The database

which stores entities is called the *entity base* and the database which stores the language which describes them is called the *knowledge base*. In logic terms, the entity base can be seen as a store of ABox elements and the knowledge base is the store of natural language vocabularies and the TBox elements.

The importing of data into the knowledge base and entity base must produce data elements which are of enough quality to enable the web services. Examples of quality aspects for the knowledge base include manual translation of vocabularies and expert validation; and for the entity base they include consistent attribute values without conflicts and correct linking between internal and external entities. To certify the quality of the imported elements, it can be useful to trace the provenance of each element. Users can see this information to appreciate the quality of the data and therefore the quality of the service built on it. This is similar to a customer that appreciates a high quality restaurant by knowing that its food ingredients are matching his standards.

1.2 The Problem

The main problem of this thesis is to increase the quality of aggregated entity stores. Data quality can have many aspects and this problem focuses only on those which we think are more relevant to the entity-centric aggregation. The problem can be divided into four major pieces of work: the quality certification of knowledge base elements, the semantic heterogeneity and the lack of explicit semantics while aggregating entities, the tracing of entity base elements with attribute values conflict avoidance and entity linking.

The first and the most important problem is the quality certification of knowledge base elements. The knowledge base is a database that stores

linguistic and domain information. These knowledge base elements are used for several high level services such as natural language processing and semantic matching. If these elements suffer from low quality characteristics, such as linguistic polysemy, then the overall quality of the services, such as the semantic matching, will be significantly reduced. This reduces also the acceptability of the whole system.

Second, the quality of the data in open data repositories is not high enough due to two reasons: the lack of explicit semantics and the semantic heterogeneity. Lack of explicit semantics is the absence of the implicit semantics that the dataset developer had in his mind while creating the dataset. If this semantics is not made explicit in the dataset itself, automated reasoning by machines will be hindered. Semantic heterogeneity appears when a dataset is developed for a specific purpose with assumptions that may not exist when the dataset is used in another public scenario.

Third, the attribute values may come from different data sources and it is important to trace the source of each attribute and to choose a strategy to handle possible conflicts. Data sources are not equal and some sources have more authority than others. A link should be provided between any pair of entities which represent the same real world entity.

1.3 The Solution

Our proposed solution is also composed of four parts which are in parallel with the division of the problem that was given in the previous section. The three parts are: the provenance of knowledge base elements, the definition of the entity import pipeline and the provenance of entity base elements.

For the knowledge base provenance, we propose a data model which is based on two main elements: the reference and the provenance. A references is used to refer to external users or resources which are represented

as entities. Provenance is used to record the source and two validators of each knowledge base element. We define also the mapping of the fields of the provenance model for each import scenario.

For the entity import, we propose a semantifying pipeline based on an entity-centric approach to enrich datasets with special focus on those datasets which originate from open data catalogs. When entities are imported, the pipeline uses a natural language processing service to enrich the text attribute values with linguistic annotations; and it uses an identity management service to assign unique identifiers to the entities. These two steps fix the absence of explicit semantics and helps in resolving semantic heterogeneity. The semantifying pipeline is built on a data cleaning tool, and its steps are divided into preliminary steps and tool steps. We describe the semantifying pipeline steps and show some user interface mock-ups for the tool steps. The required programming interface for the pipeline is also defined.

For the entity base, we propose a source tracing module that extends any existing entity-centric import process. The source tracing module contains three tools: authority, provenance and evidence. *Authority* provides rules for overriding attribute values, *provenance* specifies the source of an attribute value and *evidence* links an entity with same external entities. These three tools enable the source tracing services in the import process.

1.4 Innovative Aspects

This solution is innovative because it adds the following contributions:

- The knowledge base provenance is the first known model to capture the provenance of the language and language-independent concepts. The available provenance systems store entity-level provenance and sometimes schema-level provenance. Tracing the source of linguistic

and conceptual elements is a unique feature of our knowledge base provenance.

- The import pipeline which we use for entity-aggregation is balanced between the two extremes of current data aggregation approaches. The first approach is the Extract, Transform, Load (ETL) which is used in data warehouses and the second approach is the knowledge extraction which is used with formal knowledge systems.

- The entity base provenance handles both the provenance and the data fusion problems using authority rules in one combined model unlike other available solutions. Its definition of evidence is also different from the current 'same-as' links.

1.5 Structure of the Thesis

This thesis consists of the following chapters:

Chapter 2 presents a survey for the state-of-the-art in three research directions that are relevant to this thesis. The first section of this chapter surveys the ETL and knowledge aggregation methods and systems, the second section surveys the different approaches to handle the provenance problem. It also includes a short survey for industrial patents related to provenance.

Chapter 3 introduces the knowledge base and shows how provenance can be used to trace the source of knowledge base elements. The proposed knowledge base provenance module uses external references to users and datasets. It allows two validators to check the elements before they are finally accepted.

Chapter 4 introduces the entity base and the import pipeline that populates it. The entity base is a database that stores entities and attribute

values. These entities are imported by a pipeline which takes as input an external source. The pipeline creates or updates the entities and their attribute values.

Chapter 5 builds on the previous chapter by showing three source tracing tools: provenance, evidence and authority. We introduced these tools in the solution section above.

Chapter 6 concludes the thesis with a short summary of the work and the results. This chapter includes also the future work.

Chapter 2

State of the Art

This chapter contains a survey for the state-of-the-art in three research directions that are relevant to this thesis. In the first section, we present some ETL and knowledge aggregation methods and systems. The second section surveys the different approaches to handle the provenance problem. The chapter concludes with a short survey for patents from industry to show examples of available solutions to provenance.

2.1 ETL Tools

Importing tools can be roughly classified into general ETL systems and knowledge extraction systems. The knowledge extraction systems are more strict in the target format. The target is usually a logic system that allows some form of reasoning even if it is not implemented in the system itself. On the other hand, there is no explicit logical reasoning in generic ETL systems but they usually allow high level searching with languages like SQL.

Table 2.1 shows the description and license information for some available software for ETL. While Table 2.2 contains a list of tools used in knowledge extraction.

Name	Description
CSV2RDF4LOD [3]	A tool that produces an RDF encoding of data available in Comma Separated Values (CSV).
D2R Server [4]	A tool for publishing relational databases on the Semantic Web.
Data Master [5]	A Protege plug-in for importing schema structure and data from relational databases into Protege
Krextor [9]	An extensible XSLT-based framework for extracting RDF from XML.
MapOnto [11]	A research project aiming at discovering semantic mappings between different data models, e.g, database schemas, conceptual schemas, and ontologies.
Mapping Master [12]	A Protege-OWL plugin that can be used to transform the content of spreadsheets into OWL ontologies.
ODEMapster [14]	A NeOn plugin that provides users a Graphical User Interface that allows to create, execute, or query mappings between ontologies and databases.
OKKAM Refine extension [15]	An Open Refine extension to upload data into the OKKAM Entity Name System.
Poolparty Extractor [22]	A platform that provides precise text mining algorithms based on semantic knowledge models.
R2R [23]	A framework to allow searching the Web for mappings to reconcile vocabularies
RDF Refine [27]	An Open Refine extension that reconciles against SPARQL endpoints, RDF dumps.

RDBToOnto	RDBToOnto allows to automatically generate fine-tuned OWL ontologies from relational databases.
RDF 123 [26]	An application and web service for converting data in simple spreadsheets to an RDF graph.
RDOTE [28]	A framework for transporting data residing in RDB into the Semantic Web.
Relational.OWL [29]	Relational.OWL is a technique to automatically extract the semantics of virtually any relational databases and transform this information automatically into RDF/OWL
T2LD[32]	An automatic framework for extracting, interpreting and representing tables as Linked Data
The RDF Data Cube Vocabulary [36]	A vocabulary that allows publishing multi-dimensional data on the web in such a way that it can be linked to related data sets and concepts.
TopBraid Composer [37]	A modeling environment for developing Semantic Web ontologies and building semantic applications.
Triplify [38]	A small plugin for Web applications, which reveals the semantic structures encoded in relational databases by making database content available as RDF, JSON or Linked Data.
Virtuoso's RDF Views [40]	A tool for mapping relational data into RDF and allow the RDF representation of the relational data to be customised.

Virtuoso Sponger [39]	A middleware component of Virtuoso that generates Linked Data from a variety of data sources, and supports a wide variety of data representation and serialization formats.
VisAVis [41]	A Protege plugin that works as an intermediate layer between ontologies and relational database contents.
XLWrap [42]	A spreadsheet-to-RDF wrapper which is capable of transforming spreadsheets to arbitrary RDF graphs based on a mapping specification.

Table 2.2: A list of knowledge extraction systems

2.2 Provenance

The current approach to combine datasets from different sources is to use linked data[51]. Linked data systems face two major challenges that determine their survival[71]. These challenges are: tracing the source of data and tracing the identity of resources across datasets. The tracing of the source of data becomes critical when many datasets are imported from different data providers. A simple query to know the source of a piece of the combined data can take weeks of investigation without a suitable mechanism to describe how the combined dataset is built. The tracing of identity is the second challenge. It appears because some web resources can be treated as equivalent in some contexts. Most of the time the equivalence is expressed using the *owl:sameAs* predicate. These equivalence links provide the core linking feature of linked data.

One solution to these challenges is to create metadata vocabularies to de-

Name	Description and Features	License
Open Refine [16]	A data cleanup and transformation tool.	BSD
LODRefine [10]	LOD enabled version of Open Refine	BSD
Microsoft PowerPivot [13]	A data mash-up and data exploration tool.	Microsoft EULA
Rapid Miner [24]	A data mining and ETL tool.	AGPL
Orange [17]	A Data visualization and analysis tool.	GPL v3
Talend [34]	Data integration, data management and big data solutions.	GPL / Apache
Rattle [25]	A Graphical User Interface for Data Mining using R.	Free
Tableau [33]	Data analysis, visualization and mashup software.	Proprietary
Spotfire [31]	Business intelligence software.	Proprietary
Pentaho [20]	Open source business intelligence suite.	Apache / Commercial
CloverETL [2]	Data integration framework for transforming, cleansing and distribution of data	Commercial
Informatica Powercenter [7]	Scalable, high-performance enterprise data integration software	Commercial
Elixir Repertoire [6]	Business Intelligence suite	Commercial
Pervasive Datarush [21]	Data infrastructure software and ETL tools for big data	Commercial
Palantir [18]	platform for analyzing, integrating, and visualizing data of all kinds	Commercial
TextPipe [35]	Text mining and data conversion workbench	Commercial
Bonita Business Process Manager [1]	Business process management software.	GPL
SpagoBI [30]	An open source business intelligence suite.	Open Source
KNIME [8]	An open source data analytics, reporting and integration platform.	GPL
Palo [19]	Provides ETL services	Open Source

Table 2.1: A list of ETL tools

scribe the datasets, such as the Vocabulary of Interlinked Datasets (VoID)[1]. This vocabulary provides a model, a vocabulary of predicates and a mechanism for distributing dataset descriptions. The vocabulary captures data locations, structure and statistics. It also includes the notion of `linkset` to express the mapping between instances in different datasets. These linksets are separated from the dataset, so they can be used or updated independently from the dataset itself.

One major problem in data aggregation is the data fusion. There are three known classes of strategies in the literature for data fusion: conflict avoidance, conflict ignoring and conflict resolution (See Figure 2.1). In conflict ignoring strategies, two attribute values which are in conflict may be created in the aggregated data base which may lead to inconsistency. In conflict resolution strategies, the conflict between two or more attribute values is resolved by taking their average for instance.

Strategy	Classification	Short Description
PASS IT ON	ignoring	escalates conflicts to user or application
CONSIDER ALL POSSIBILITIES	ignoring	creates all possible value combinations
TAKE THE INFORMATION	avoiding, instance based	prefers values over null values
NO GOSSIPING	avoiding, instance based	returns only consistent tuples
TRUST YOUR FRIENDS	avoiding, metadata based	takes the value of a preferred source
CRY WITH THE WOLVES	resolution, instance based, deciding	takes the most often occurring value
ROLL THE DICE	resolution, instance based, deciding	takes a random value
MEET IN THE MIDDLE	resolution, instance based, mediating	takes an average value
KEEP UP TO DATE	resolution, metadata based, deciding	takes the most recent value

Figure 2.1: Possible Conflict Handling Strategies from [53]

Finally, as a part of our state-of-the-art survey for the latest advances in provenance, we present a summary of the provenance week 2014. The provenance week is a combined event of two conferences IPAW 2014 and TAPP 2014.

- Initial workshops

[1] www.w3.org/TR/void/

The first workshop presented the PROV standard with recipes and tools. The tools include provenance translator that converts between different serializations formats such as PROV-N and PROV-XML, provenance validator that checks the provenance against PROV constrains, provenance store and provenance online editor. Finally a hands on session was given about ProvStore and Git2Prov tools.

Then a session on new scenarios of provenance presented four applications in the domains of scientific research, cloud computing, geospatial data and climate research: [59] is a tool for gathering of provenance information about the datasets used in research papers. [44] understands the behavior of Google Cloud from its one month log data. [65] describes user requirements for modeling geospatial provenance. [86] defines a formal provenance representation for global climate change data.

Four more applications were presented in the next session: [91] studies query generation for PROV-O data. [76] shows how to use provenance for online decision making with a crowd sourcing scenario. [74] is an application-independent methodology for analyzing data based on the network metrics of provenance graphs for the assessment of data quality in an automated manner. [78] uses provenance to optimize the parallel execution of scientific workflows.

- IPAW

  The workshop discussed these topics: standardization, application, architecture, security and reproducibility.

  In the standardization discussion, two papers were presented: PROV Abs [79], a tool for interactive experimentation with policies and abstractions for sharing sensitive provenance data. PROV Gen [61] produces large synthetic provenance graphs with predictable properties

and of arbitrary size. Synthetic provenance graphs are useful for testing and experimenting.

In the application discussion, three applications were presented: [77] is an application for Prov PingBack to interconnect provenance records that would traditionally sit in isolation, [75] proposes a generic and application-independent approach to interpret provenance of data to make online decisions and [67] collects compile-time and run-time provenance for benchmarking systems.

In the architecture discussion, three papers were presented: noWorkflow [80] is a tool that transparently captures provenance of scripts and enables reproducibility, LabelFlow [46] uses domain-specific labels with workflow provenance as a platform for data artifacts labelling; and [84] proposes software provenance to be included as part of software packages.

In the security discussion, three papers were presented: [56] is a survey for security in seven provenance systems, [45] proposes a policy control framework for integrating and developing services that generate and use provenance. [63] provides security controls for protecting provenance information.

Finally in the reproducibility discussion, three papers were presented: [85] is an approach to capturing data provenance based on taint tracking. [89] generates electronic notebook documentation from multi-environment workflows by using provenance represented in the W3C PROV model. [58] describes a provenance computation technique that uses the structure of workflow specifications to pre-compute the access paths of all dependent data items prior to workflow execution.

- Demos and Posters

  Four demos were presented: [81] is an implementation of provenance

query service with PingBack following Provenance Access and Query (PROV-AQ) standard. [50] presented a system that shows information about Internet of Things devices to users. [73] was about Prov-O-Viz1, an online web-based visualization for provenance given in PROV-O. Finally [57] presented PBase, a provenance repository that supports search with ranking capability.

Several problems where discussed in the poster session. These problems include using provenance in query answering by limiting the sources that can be used to answer a query, making provenance interoperable, adding provenance to an RDF store, tracing provenance of software development as a quality factor, abstracting provenance to hide private details.

Two provenance stores were presented, one provides a Web graphical user interface for visualizing provenance graphs, while the other is a public repository that allows user and applications to publish their provenance on the web. Another two solutions extended the PROV-DM and PROV-O standards to allow provenance of sensor web and social computing respectively.

Few posters presented the use of provenance in several domains: medical, weather, agriculture, geospatial and taxonomy alignment scenarios. Other posters discussed engineering choices for provenance. One engineering solution was proposing to use aspect-oriented programming to seamlessly integrate provenance.

2.3 Patents

The patent archives include several approaches for treating the provenance problem. For instance, the University Of Southern California proposed a system and method for data provenance in the case of multi-

ple, related workflows [90]. Researchers from the University Of Utah Research Foundation proposed provenance management system for pre-existing applications[62]. IBM corporation shows a system to use and enforce provenance constrains which calculates provenance based on data or meta-data changes and determines whether the calculated provenance violates the constraints[87]. Microsoft corporation used provenance in health care domain[88]. Other researches produces a system to visualize provenance to improve search query[48]. It can be easily noted that these provenance tracing approaches are either too generic and they are not tailored to a specific work-flow or they are too specific for one domain such as the medical domain.

Chapter 3

Knowledge Base Provenance

The knowledge base (KB) is a database for background knowledge. The background knowledge must satisfy some quality requirements before it can be used by semantic services. In this chapter we present how we certify the quality of the knowledge in the KB.

First we present the quality certification problem, then we list the elements of the KB and how they are imported or created. Next we present our proposed data model which is based on two main elements: the *reference* and the *provenance*. A references is used to refer to external users or resources which are represented as entities, therefore we show also the definition of user and resource entities with the reference model. Provenance is used to record the source and the validators of the KB element, therefore we show also the mapping of the provenance fields for each import scenario with the provenance model. Then we show the Web API model and services for KB provenance. Finally, we present few use cases that were treated with the proposed model.

3.1 The Problem

The background knowledge is required to build semantic services [68]. The quality of a semantic service depends largely on the quality of the back

ground knowledge behind it. If the quality of the knowledge is low, the semantic services which are built over it will fail to deliver the required results. For instance, if the background knowledge suffers from high polysemy (one natural language word that has more than one meaning), it will be less useful for the natural language processing and semantic services [43].

Data quality is generally defined as "fitness for use". The KB is used in the context of semantic services, hence the fitness here can be specified as correctness, completeness and duplication-free. Correctness of the knowledge base is the right definition of each KB element and its relations with other elements. Completeness of the knowledge base is the coverage of the elements in one language and the coverage of all the languages required for the specific application. Duplication-free is the avoidance of repeated elements in the knowledge base, such as duplicated concepts in the concept graph or duplicated meaning for one word.

With this definition of data quality, it can be noted that the quality of knowledge base is set at the elements creation time. This is because the creation of a KB element causes either an increase or a decrease of the knowledge base quality. For instance, creating a correct example for a synset increases the quality, while creating a wrong translation for a word decreases the quality. Since the elements creation time is when the quality is changed, the quality can be certified by tracing the sources and users who participated in the creation of the KB element.

Although several approaches for provenance exist in the literature as presented in Chapter 2, they either discussed the problem of provenance at the instance or the schema levels. There is no existing solution that fulfills the need for provenance at the language and conceptual level, therefore we developed this novel provenance model as a quality certification tool for back ground knowledge bases of semantic web services.

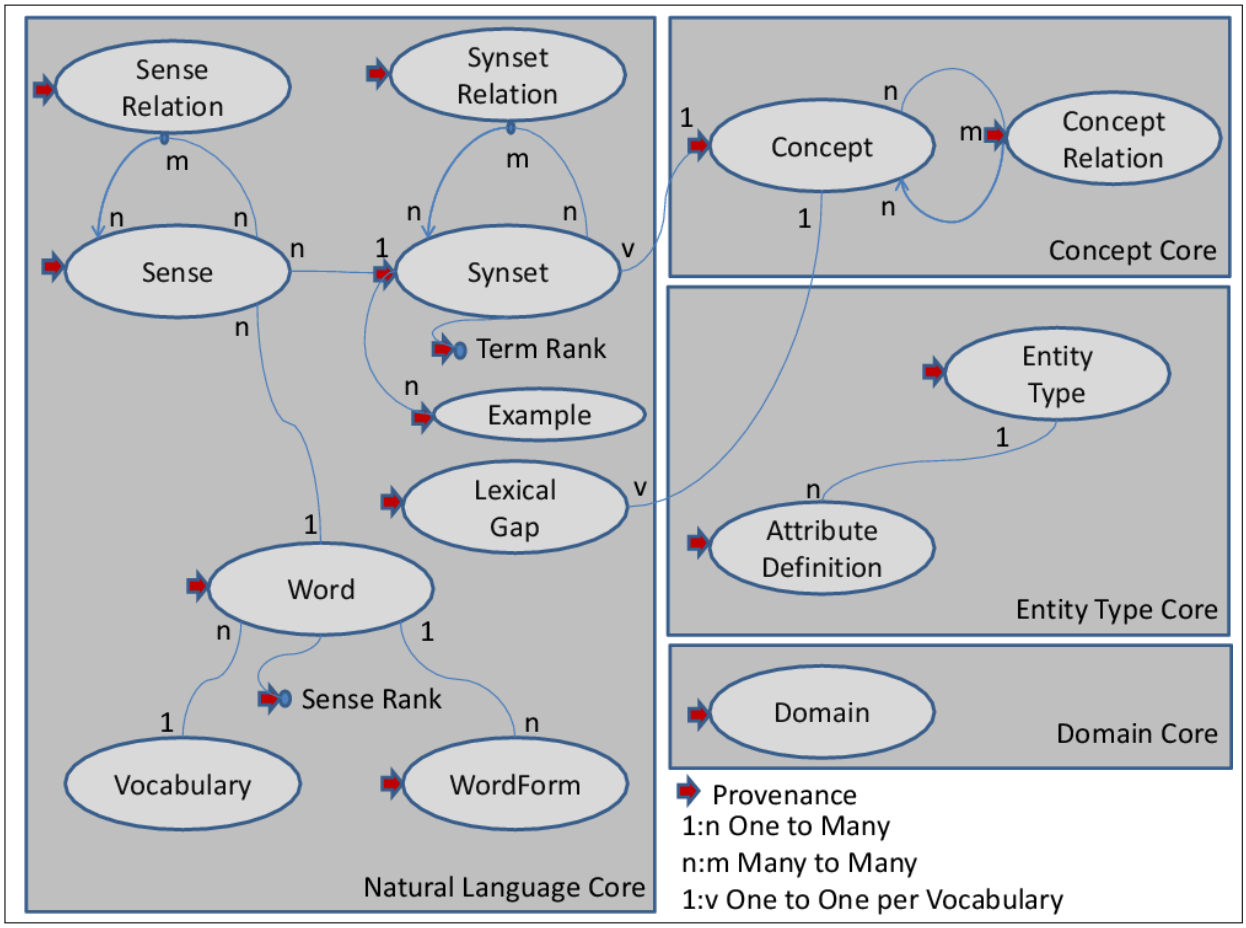


Figure 3.1: Knowledge Base Elements

3.2 UKC Elements

The universal knowledge core (UKC) is a central background knowledge database which stores several types of elements. These elements are called knowledge base (KB) elements and some of them are traced by provenance. The KB elements are classified into four groups: *Natural Language Core*, *Concept Core*, *Entity Type Core* and *Domain Core*. Figure 3.1 shows the KB elements and their inter-relations. It also show that provenance can be assigned to:

1. Entity Type: a sort of template including the definition of the at-

tributes and relations allowed.

2. Attribute Definition: the name and data type of the attribute.

3. Concept: a formal notion denoting a Class, an Attribute Name (including Relation names) or an Attribute Value.

4. Concept Relation: a connection between related concepts.

5. Domain: a set of entity types and a subset of their attributes which determine the terminology of the domain.

6. Lexical Gap: a concept that cannot be lexicalized in the language.

7. Word: represents a word that belongs to a specific vocabulary.

8. Wordform: represents a derived form of a given word.

9. Sense: represents a word sense, basically defining the link between a word and a synset this word belongs to.

10. Synset: a group of words with same sense, i.e. which are synonyms in a language.

11. Sense Relation: a connection between related senses.

12. Synset Relation: a connection between related synsets.

13. Synset Example: an example attached to a specific synset

14. Word Sense Rank: the word sense rank of the word sense (used to order the senses of a word)

15. Synset Word Rank: the concept word rank of the word sense (used to order the senses of a concepts)

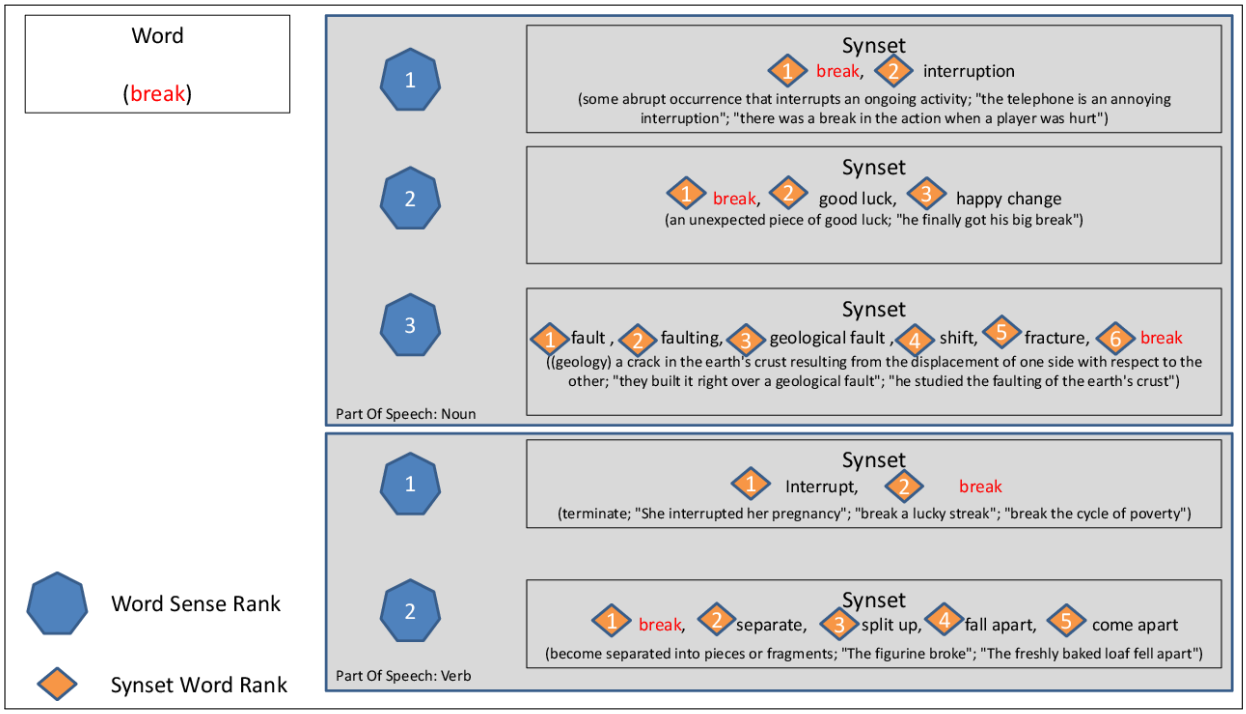


Figure 3.2: An example for word sense ranks and synset word ranks

The last two items, namely word sense rank and synset word rank, are not KB elements. They are two ranks associated with KB elements as shown in Figure 3.2. This figure shows an example for the natural language word *break*. Since this word can be a noun or a verb, it has a group of synsets for each part of speech. In each part of speech group, the synsets are ordered with the *Word Sense Rank* which is shown in hexagons. In each synset, the senses are ordered with the *Synset Word Rank* which is shown in squares.

The word sense rank is associated with a word and a part of speech for this word. The synset word rank is associated with a synset. The provenance is specified for the whole rank and not for a single entry in the rank.

3.3 References

A reference is data structure used to provide external references. There are three reference types:

1. UserReference. Refers to a user who participated in the import and his editorial role.
2. ResourceReference. Refers to an external resource.
3. ResourcePartReference. Refers to a part of an external resource. (e.g. a row in a table)

Listing 3.1: Reference BNF for KB Provenance

```
Reference        ::=    User
                        | Resource
                        | ResourcePart

User             ::=    Id
                        , EditorialRole

EditorialRole    ::=    String

Resource         ::=    Id
                        , {ResourcePart}*

ResourcePart     ::=    Identifier
                        , URI

Identifier       ::=    Name
                        ,Value
```

```
Name        ::=    String
Value       ::=    String
URI         ::=    String
Id          ::=    Long
```

Where:

- Reference: is an external reference to a user, a resource or a resource part.
- User: is a reference to a user with the editorial role assigned to this user. The user is represented as an entity with a set of attributes defined in Section 3.3.1.
- Resource: is a reference to an external resource. The resource is represented as an entity with a set of attributes defined in Section 3.3.1.
- ResourcePart: is a reference to a part in an external resource.
- Identifier: is the identifier of the part in the external resource.
- Name: is the name of the identifier of the resource part. [e.g., synset_offset]
- Value: is the value of the identifier of the resource part. [e.g., n#00145679]
- URI: is the universal resource identifier of the resource part. [e.g. wordnet/2.1/dict/data.noun]
- Id: is the internal identifier of a resource or an entity in an entity base. More information on the resource and user entities are given in the Section 3.3.1.
- EditorialRole: is the editorial role that can be assigned to a user.

The editorial role depends on the importing scenario. In the UKC 1.0 first-time bootstrapping scenario, there are two predefined editorial roles UKC1.0 IMPORTER and UKC1.0 VALIDATOR. In the language development scenario, the editorial role encodes the name of the LKC (NAME), the source language (L1) and the target language (L2). The possible editorial roles for the LKC follow this regular expression:

[< NAME >_](L1 | L2)_(IMPORTER | MANAGER)
[< NAME >_](L1–L2 | L2–L1)_(DEVELOPER | VALIDATOR)

The first optional part [NAME_] which has the LKC name is added after copying (promoting) the knowledge base element and its provenance from the LKC to the UKC.

3.3.1 Modeling Users and Resources

We present here the minimal set of attributes for modeling user and resource entities. The general attributes of an Entity are given in Table 3.1. These attributes are inherited by both the user and resource entities. In this table (and also in the next tables) The first column shows the attribute, the second column shows the data type and the third column shows the reference dataset which gave the definition of the attribute. The <P >symbol beside a data type means that the corresponding attribute is permanent (i.e. it is not constrained with a time validity, but it is always valid).

Attribute	Data type	Reference Dataset
Name	NLString []	
Class	Concept <P >	
SURL	String <P >	UKC system itself

Table 3.1: Entity attributes

- 1. Person

- Category: personal

Attribute	**Data type**	**Reference Dataset**
Name	NLString []	Person herself
Class	Concept	UKC system itself
Gender	ENUM(MALE, FEMALE) <P>	person herself
birth Bate	Date <P>	Person herself
Email	String	Person herself

Relation	**Entity type**	**Reference Dataset**
Country of citizenship	Country	Person herself
Country where hiving now	Country	Person herself
City where living now	City	Person herself
Photo	String	Person herself

- Category: biography

Attribute	**Data type**	**Reference Dataset**
Degree	ENUM(PhD, MASTERS, BACHELOR, HIGH SCHOOL, NONE) [] <P>	Person herself (Note that with array symbol we mean that a person ctn select more than one degree)
Work	Boolean	Person herself
Student	Boolean	Person herself

- Category: user

Attribute	**Data type**	**Reference Dataset**
Login name	String	Person herself
Password	String	Person herself
Joining date	Date	Person herself
Leaving date	Date	Person herself

Relation	**Entity type**	**Reference Dataset**
Recommender	Person <P>	UKC existing user (possibly suggested by person herself)

- Category: expertise

Attribute	**Data type**	**Reference Dataset**
Language proficiency	<Language, Level> []	Person herself
• Level	ENUM(A1, A2, B1, B2, C1, C2)	person herself

Relation	**Entity type**	**Reference dataset**
Language	Language	Person herself

- 2. Resource

- Category: general

Attribute	Data type	Reference Dataset
Name	NLString []	Entity importer herself
Class	Concept <P>	Entity importer herself
description	SString <P>	Entity importer herself
License	String	Entity importer herself
Note	NLString <P>	Entity importer herself
Version	String <P>	Entity importer herself
Release	String <P>	Entity importer herself
Date of publication	Date <P>	Entity importer herself

Relation	Entity type	Reference Dataset
Owner	Person and/or Organization	Entity importer herself

- Category: resource identity

Attribute	Data type	Reference Dataset
Homepage URL	String	Entity importer herself
Resource URL	String	Entity importer herself

Attribute	Structured type	Reference Dataset
KiDF URL	KiDF URL	Entity importer herself

- Structured types

KiDF URL

Category: recourse identity

Atbritute	**Data type**	**Reference dataset**
Name	NLString []	Entity importer herself
Description	SString	Entity importer herself
Note	NLString	Entity importer herself
URL	String	Entity importer herself

3.4 Provenance

After defining the references, we present the definition of the provenance in the figure below.

Listing 3.2: Provenance BNF

```
Provenance        ::=   Provenance_ID
                        , {KB_ID, Note}*
                        , Source
                        [, Validator1]
                        [, Validator2]
                        [, ProvenanceNote]
Provenance_ID     ::=   Long
KB_ID             ::=   Long
Note              ::=   String
ProvenanceNote    ::=   String
Source            ::=   Reference
Validator1        ::=   User
Validator2        ::=   User
```

Where:

- Provenance: is a representation of the source, the first validator and the second validator that participated in the creation of (zero or more) knowledge base elements.
- Provenance_ID: is a unique identifier for the KB Provenance which is generated and managed by the provenance framework.
- KB_ID: is the unique identifier of the knowledge base element to which the provenance is related. It is generated and managed internally by the knowledge base.

- Source: is a reference to an external source as defined in Listing 3.1. It can be a resource or a user. See Table 3.2 for the content of this field in each import scenario.

- Validator1: is a reference to an external user defined in Listing 3.1. The user is either the importer who imported from the source or the language validator who validated an element from the source. See Table 3.2 for the content of this field in each import scenario.

- Validator2: is a reference to an external user who performed the final validation on the imported element as defined in Listing 3.1. See Table 3.2 for the content of this field in each import scenario.

- Note: is a string that is used to record user notes on the knowledge base element.

- ProvenanceNote: is a string that is used to record user notes on the provenance. This note is written only by validator2.

This provenance model implements a provenance graph (see Figure 3.3) between the knowledge base elements and their external sources. All objects of the provenance graph in this figure are presented before except ProvenanceElement. This object is introduced to allow the provenance graph to be implemented as a separate database without changing the knowledge base.

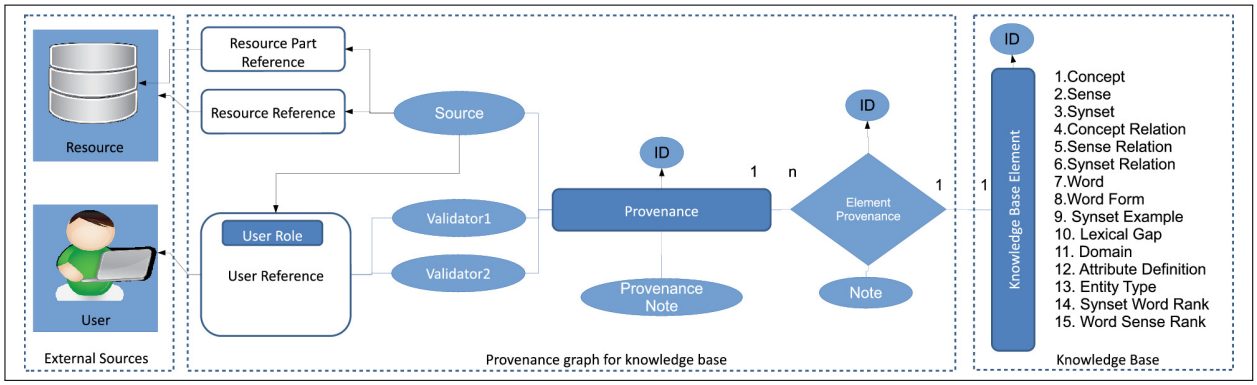


Figure 3.3: Provenance graph for the knowledge base

3.4.1 Knowledge import scenarios

Provenance is used to trace the KB elements at their creation time. The KB elements are created in the universal knowledge core (UKC) through import scenarios, such as importing from an external dataset or developing a translation of a vocabulary by an expert user. In addition to the UKC, there are three other types of knowledge cores which are involved in the import scenarios. These are the Peer Knowledge Core (PKC), Language Knowledge Core (LKC) and Domain Knowledge Core (DKC).

The PKC is a knowledge core which runs on a peer to provide services that support an application. The LKC is a special PKC used to develop languages; and the DKC is a special PKC used to develop domains. Each LKC has a name, a source language and a target language. In all import scenarios, elements are not created directly in the UKC. They are first imported in an intermediate knowledge base (PKC, LKC or DKC) then promoted to the UKC. The import scenarios are shown in Figure 3.4.

The import operation is different from other operations that transfer KB elements between knowledge cores. In the import operation, the elements are created from an external source (a user or a resource), while in transfer operations between knowledge cores the elements are copied from a PKC,

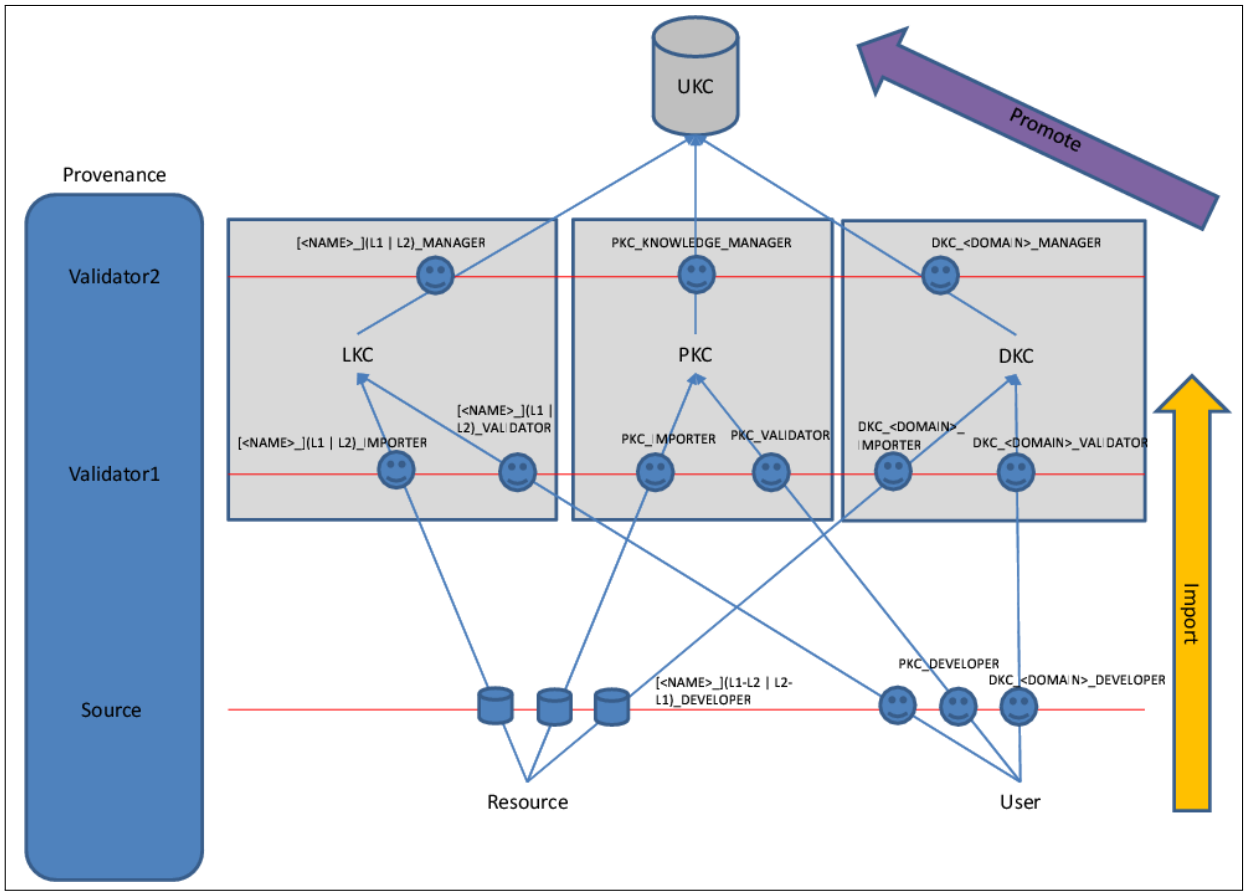


Figure 3.4: UKC importing scenarios

LKC, DKC or UKC. No change is required in provenance when copying an element within the system between the knowledge cores; except that after copying an element into the UKC from an intermediate knowledge base (such as a LKC), the name of the intermediate knowledge base should be added to the provenance.

Provenance is specified only in the first time creation from an external source. This operation is called Importing operation. Provenance is not required when elements are copied from PKC, LKC or DKC into the UKC. This operation is called Promotion operation and it just copies the provenance of the element and adds the intermediate knowledge base when necessary.

Table 3.2 shows how the fields of the provenance are filled in each import scenario.

Import Scenario		Required Provenance		
Source	Intermediate KB	Source	Validator1	Validator2
Resource	LKC	RESOURCE	[<NAME>_](L1 | L2)_IMPORTER	[<NAME>_](L1 | L2)_MANAGER
	PKC	RESOURCE	PKC_IMPORTER	PKC_KNOWLEDGE_MANAGER
	DKC	RESOURCE	DKC_<DOMAIN>_IMPORTER	DKC_<DOMAIN>_MANAGER
User	LKC	[<NAME>_](L1-L2 | L2-L1)_DEVELOPER	[<NAME>_](L1 | L2)_VALIDATOR	[<NAME>_](L1 | L2)_MANAGER
	PKC	PKC_DEVELOPER	PKC_VALIDATOR	PKC_KNOWLEDGE_MANAGER
	DKC	DKC_<DOMAIN>_DEVELOPER	DKC_<DOMAIN>_VALIDATOR	DKC_<DOMAIN>_MANAGER

Table 3.2: Required provenance for each import scenario

3.5 Web API

We implemented a web API for provenance to make it more usable for web scenarios. This was necessary because the main applications for our provenance are web-based. The web API is divided into a model and a list of services. We used JSON as the serialization format of the model and we followed REST principles in designing the services. In this section, we present the data model and the services of the Web API component for knowledge base provenance.

3.5.1 Data Model

The data model of the knowledge base provenance web API has three classes as shown in 3.5. The classes are *KBReference* which represents both the user and resource references; *KBProvenance* which represents the provenance tuple with the source, validator1, validator2 and the provenance note; and finally *KBElementProvenance* which provides the link between the provenance and the knowledge base element that is identified with the element ID (and the part of speech in the case of word sense ranks).

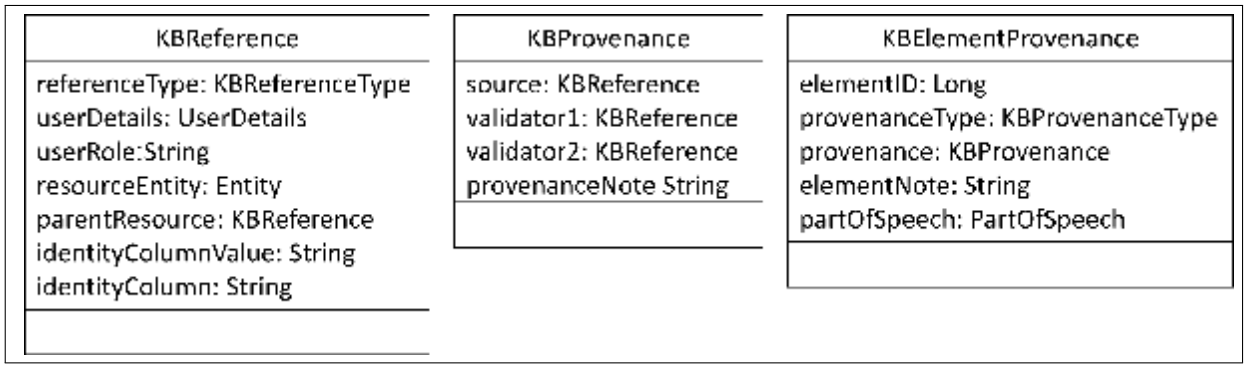


Figure 3.5: UML for Knowledge Base Provenance Web API Model

3.5.2 Services

A list of web services is developed for accessing the provenance Web data model. The service list is shown in Table 3.3. The first column shows the HTTP method, the second column shows the end point and the third column shows the description of this end point. The services provide read and write operations for the model objects. The read operations include searching such as finding a provenance by one or more of its fields (source, validator1 or validator2). The services are designed following the REST principles to ensure scalability and usability in modern Web applications.

HTTP Method	End point	Description
GET	/kb/references/{id}	reads a reference by its ID
POST	/kb/references/	creates a reference
PUT	/kb/references/{id}	updates a reference by its ID
DELETE	/kb/references/{id}	deletes a reference by its ID
GET	/kb/provenance/{id}	reads a provenance by its ID
GET	/kb/provenance/source={1} &validator1={2}&validator2={3}	Reads provenance(s) by source, validator1 and validator2
POST	/kb/provenance/	creates a provenance
PUT	/kb/provenance/{id}	updates a provenance by its ID
DELETE	/kb/provenance/{id}	deletes a provenance by its ID
GET	/kb/elementprovenance/{id} /elementType={1}&elementID={2}	reads an element provenance by its ID, or by the element type and the element ID
POST	/kb/elementprovenance/	creates a element provenance
PUT	/kb/elementprovenance/{id}	updates a element provenance by its ID
DELETE	/kb/elementprovenance/{id}	deletes a element provenance by its ID

Table 3.3: List of web services for knowledge base provenance

3.6 User Interface

The knowledge base provenance can be visualized within the knowledge base element user interface. As shown in Figure 3.6, this basic user interface shows the source, validator1, validator2 and a note for an element. More complex user interface can be developed to query the provenance and the elements which are related to it.

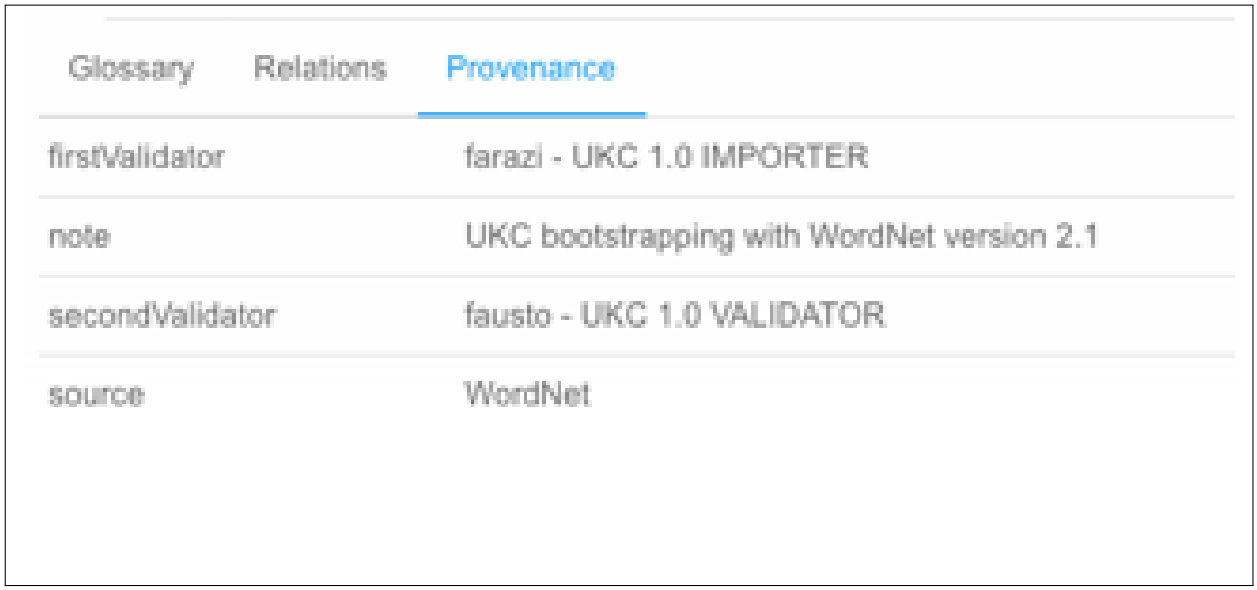


Figure 3.6: Basic visualization for knowledge base provenance

3.7 Use Cases

The knowledge base provenance was validated with few use cases from the Universal Knowledge Core project. This section presents three use cases which used the proposed provenance successfully to ensure high quality for the knowledge elements. The three cases are for importing the English, Chinese and Mongolian vocabularies. English is a basic language for the background knowledge due to its global use. Chinese is the most spoken language and it is one of the most used languages on the Web. Mongolian is not widely used as English and Chinese, therefore it provides a different and interesting use case.

3.7.1 Provenance for English Vocabulary

In addition to the six scenarios which were presented in Figure 3.4, there is one extra scenario which is the importing of English WordNet for UKC 1.0.

This scenario is presented in Figure 3.7. For this scenario, we added the following user roles: UKC_VALIDATOR and KNOWLEDGE_IMPORTER to the definition of reference in Figure 3.1.

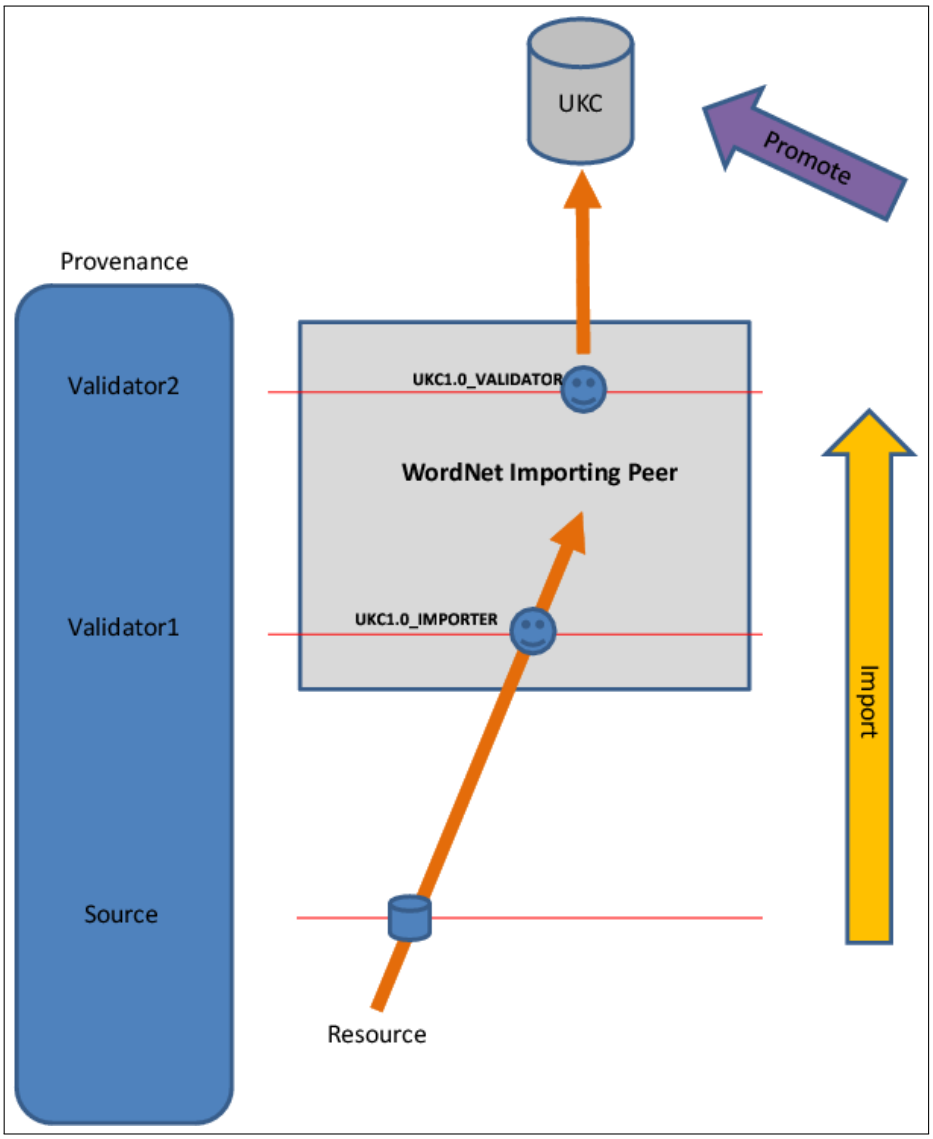


Figure 3.7: importing English WordNet for UKC 1.0

Import Scenario		Required Provenance		
Source	Intermediate KB	Source	Validator1	Validator2
Resource	PKC	Resource (WordNet)	UKC1.0_IMPORTER	UKC1.0_VADILATOR

Table 3.4: Required provenance for importing UKC 1.0

As shown in Figure 3.7, the English vocabulary was imported on a special importing peer which we called here *WordNet Importing Peer*. The source of this import is the WordNet, the validator1 is the importer in the role of UKC1.0_IMPORTER and the validator2 is the UKC validator in the role of UKC1.0_VALIDATOR. This information is presented again in Table 3.4.

This use case is important because English is a basic language for the universal knowledge core due to its universal use. It is used as a source language in many translating scenarios. English WordNet is also the first and the most developed linguistic resource for the semantic web. Similar linguistic resources for other languages (i.e. other WordNets) follow the same data structures and sometimes they reuse the same identifiers for common elements. The list of elements that are imported from the English WordNet are given in Table 3.5.

Element	Count
Word	133973
Synset	109941
Sense	191522
Sense Relation	47691
Synset Relation	12813
Word Form	4718
Synset Example	48459

Table 3.5: KB Elements imported from English WordNet

3.7.2 Provenance for Chinese Vocabulary

The development of Chinese language in the Language Knowledge Core (LKC) is done firstly by translating the space domain from English to Chinese in order to study the mapping between English and Chinese synsets and then by integrating a Chinese WordNet.

Space domain is translated from English to Chinese. The space domain[69] is a set of knowledge base elements (concepts, words, synsets and their relations) which are used to describe locations in space. Examples for concepts that can be found in space domain are: *City* and *Mountain*.

The Chinese WordNet (simplified version) is a linguistic resource for Chinese language that was developed by Southeast University of China. It contains translations for the Chinese words of synsets, but, it is out of Chinese glosses. However, it aligned with English glosses which came from English WordNet. I.e. it reuses the same identifiers from the original English WordNet for synsets, therefore it was straightforward to link the imported synsets from Chinese WordNet with the corresponding concepts in the universal knowledge core.

The result of the Chinese language development is shown in Table 3.6. The table shows the KB element in the first column, the number of imported elements in the second column and the source of the element in the third column. These elements are now ready to be used in Chinese semantic web applications. With each imported knowledge element, the provenance is added to record who is the importer, the translator and the validator.

Element	Count	Source
Word	89780	Chinese Wordnet
Synset	96636	Chinese Wordnet
Sense	119959	Chinese Wordnet
Lexical Gap	21	Translator
Word Sense Rank	119959	-
Synset Word Rank	119959	Chinese Wordnet

Table 3.6: KB Elements in Chinese importing experiment

3.7.3 Provenance for Mongolian Vocabulary

The last use case that we present here is an experiment of ontology localization [47]. The experiment was done on about 1000 English concepts which were originally developed in English and later they are translated into Mongolian. The concepts were taken from space ontology. This use case is interesting because there was no available resources to use as a background knowledge for the Mongolian language. Starting from the concepts which are developed for English language, it was possible to build a Mongolian resource for the space domain. A manual approach was used in this Mongolian ontology localization for translation and validation. The approach has two phases which required provenance: the concept translation and the concept addition.

In concept translation phase, the language translator may develop a word, a synset, a lexical gap or a concept. For each element created, a new provenance will be generated with the source referring to the translator. If a language validator confirms an already developed element in this phase, then the provenance of that element is updated by linking validator1 to the instance of the UserReference which corresponds to the user name with the role of the language validator that is LKC_Validator. As soon as the UKC validator confirms a validated element, the provenance of that element is updated with the instantiation of the attribute validator2 referring to the name and role of the validator (i.e., UKC_Validator) of the given context. This marks the element as completely validated and accepted.

In the concept addition phase, the language translator may create a new concept and its related lexical components such as synset, word, etc. in the target language and optionally in the source language. In this case the provenance source for each of the objects will be instantiated with the translator for her LKC developer role. Again in this phase if the

LKC English validator evaluates the source language synset provided by the language translator, the provenance validator1 is instantiated with her LKC_Validator role. If it happens that the LKC English validator translates back the new concept into the source language, in this case the source is filled in with her role as LKC Developer and the validator1 is left uninstantiated. Similarly to the concept translation, a UKC validator checks the correctness of the concept addition and she becomes validator2 with the corresponding role.

3.8 Conclusions

In this chapter, we presented the problem of quality certification for a back ground knowledge base in semantic web scenarios. The knowledge base elements are imported from external sources which can be users or resources. Two expert validators check the elements before they are finally accepted. We developed a model for tracing the provenance of this importing and validation process. The model has been used successfully in few projects.

Our experiments show that tracing the creation provenance guarantees a good level of quality for the whole knowledge base. This in turn increases the usability of the semantic services built on top of the knowledge base. The provenance solution that we presented in this chapter can be used as a tool for enforcing and tracing quality for any similar back ground knowledge.

Some parts of this chapter have been published in [64].

Chapter 4

Entity Base

4.1 Introduction to the entity base

The aggregation approach that we follow is entity-centric. It is rooted in the DERA methodology[68] which captures knowledge with four facets: Domains, Entities, Relations and Attributes. The canonical and universal entity base is Entitypedia[1]. The unique features of the entity base are (1) the separation between the natural language and formal language; (2) the separation between the instances, the types and the concepts in the formal language part.

[1] http://entitypedia.org/

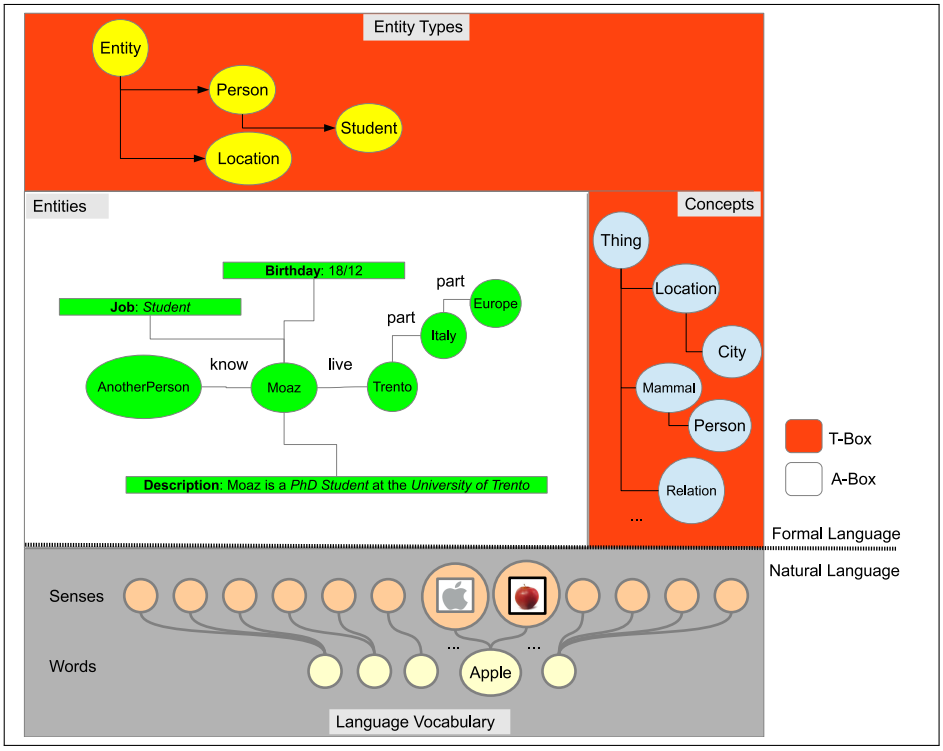


Figure 4.1: Logical separation of elements in our entity-centric model

Figure 4.1 shows a high-level view of the entity-centric model. It divides the elements in two parts: natural language part and formal language part. The formal language part is also divided into two parts: the A-Box and the T-Box. The A-Box part is the entities part and the T-Box part is divided between the Concepts and Entity Types. The result of this division is four parts:

- The natural language part is the language vocabulary which is used to describe natural language words and their meaning (sense). A natural language word may refer to more than one meaning (e.g. the shown English word *Apple* can refer to the fruit or to the computer company), one meaning can be denoted by different words or one word can have exactly one meaning. The vocabulary part stores more elements about the natural language in addition to words and senses

as shown in Chapter 3.

- The concept part is a graph of concepts where the links between the concepts are concept relations. Concepts are natural language independent.
- The entity types part is a type hierarchy with a list of attribute definitions for each type. This part acts like the schema for the entities part.
- The entities part is the instances part where entity types and attribute definitions can be instantiated.

The four parts are connected together with links which are not shown in Figure 4.1 to avoid confusion.

The entities part (i.e. the A-Box) is stored in the entity base while the other three parts (i.e. the T-Box and the natural language vocabulary) are stored in the knowledge base. This logical separation allows the system to make the hidden semantic information explicit which enables many services such as entity search with semantics. It also has other benefits such as the multi-language feature.

We refer to real life objects that are of enough importance to be given a name as entities. Examples for entities are *Italy* and *Barack Obama*. There are different types of entities, such as Locations and Persons. Italy is an entity that has a type of *Location* and *Barack Obama* is an entity that has a type of *Person*. The type of entity gives the list of definitions for the attributes that can be assigned to an entity of this type. Location entities may have the attribute *Area* which holds the value of the total area of the location, and Person entities may have the attribute *Birthdate* which holds the date of the birth for this person.

The entity-centric approach is not limited to physical entities like people and locations, but it is used also to model nonphysical ones such as

intellectual and digital objects. Books and movies are two examples of intellectual entities; Computer files and digital pictures are two examples of digital entities.

The type of entity gives the list of attribute definitions that can be assigned to an entity of this type. Location entities may have the attribute Area which holds the value of the total area of the location, and Person entities may have the attribute Birthdate which holds the date of the birth for this person. The values of the attribute definitions for a specific entity are called attribute values. An attribute definition can be single-valued (e.g. the birthdate attribute of a person), or it can be multi-valued (e.g. the email attribute of a person).

An entity is defined as:

Listing 4.1: Entity BNF

```
Entity ::= < {EntityName}+,
             {AttributeName,
              AttributeValue}* >
```

Where:

- EntityName is the name of the entity. An entity can have one or more names.
- AttributeName is the name of an attribute which is associated with the entity type of this entity.
- AttributeValue is a value of an attribute which is defined by the AttributeName.

For example, an EntityName can be *Barak Obama*; and an AttributeValue can be *August 4, 1961* which is the value of the birthdate attribute of this person.

An entity base supports the following operations on entities and their attributes:

- CreateEntity(Entity e): creates a new entity in the entity base.
- CreateAttributeValue(AttribueDefinition ad, AttribueValue av): creates a new attribute value for an attribute definition in the entity base.
- UpdateAttributeValue(AttribueDefinition ad, AttribueValue av): updates an existing attribute value in the entity base.
- GetEntityIdentifier(Entity e): returns the unique identifier of an entity in the entity base or null if the entity does not exist.
- ReadAttributeValue(AttribueDefinition ad): reads the attribute value for the attribute definition ad.
- IsMultiValuedAttribue(AttribueDefinition ad): returns true if the given attribute definition is a multi-valued attribute.

4.1.1 The semantic interface

We present here a summary for the semantic services which are exposed by the knowledge base and entity base. The semantic services are accessed through programming interfaces. These interfaces allow the service to have different implementations.

The semantic services are:

1. Semantic Matching service is used to perform schema matching.
2. Entity Type service is used to perform read entity types or to add attribute definitions and unique indexes to them.

3. Knowledge service is used to access vocabularies to search for natural language words.
4. NLP service is used to provide natural language processing services such as word sense disambiguation, entity disambiguation and named entity recognition.
5. Entity service is used for CRUD on entities and attributes. Operations that can be performed with the entity service were defined in the previous section.
6. Identity service is used for creating unique identifiers for new entities.
7. Search service is used to find entities that match a given query.

4.2 Problem

Creating a new catalog for open data requires preparing data for public use if the data was not prepared from the design and the implementation for this public scenario. While it could be easy for human users to understand or guess the semantics behind the data, automatic reasoning by machines will be hindered by two problems known as lack of explicit semantics and semantic heterogeneity.

I. Lack of explicit semantics is the absence of the semantics that the dataset developer had in his mind while creating the dataset. This implicit semantics gives the meaning of the vocabularies and values used in the dataset. When the dataset is published in a catalog without this semantics made explicit with the dataset, automatic reasoning by machines will be hard. Better services can be enabled by resolving the lack of explicit semantics[83].

For instance, let us assume a government dataset that contains data about the city of Trento in Italy. There is an assumption made by the data

author that Trento is the city in North Italy. It is possible that another location in the world may have the same name, for instance Trento in the province of Agusan del Sur, Philippines. It is clear which city is referred to in the data only when the semantics are made explicit by assigning a unique identifier to the Trento city which is in North Italy and link the city with that identifier.

II. Semantic heterogeneity refers to differences or similarities in the meaning of local data[72]. Other kinds of heterogeneity are structural (schema), syntactic (format) heterogeneity [66]. Heterogeneity is caused by autonomously designed and developed datasets. This heterogeneity makes it harder to link data from different sources together and complicates the development of services.

For instance, let us assume again a government dataset that contains data about the city of Trento and another dataset from a public knowledge base like Wikipedia about Trento. Each dataset represents the same city of Trento with different schema, with different format and with different meaning than the other dataset. The government dataset may have the attribute inhabitants that holds the number of people living in the city in thousands, while the Wikipedia dataset may have the attribute Population that holds the number of people living in the city not in the format of thousands.

4.3 Import Pipeline

The entity base stores entities from external datasets by importing them. This section presents the import pipeline which semantifies the datasets and extracts entities from them. The pipeline is composed of steps that are divided into preliminary and tool steps. The preliminary steps are executed before processing the dataset. The tool steps are steps which are

executed using a graphical user interface that helps the user to semantify the given datasets.

4.3.1 Catalog Importing Workflow

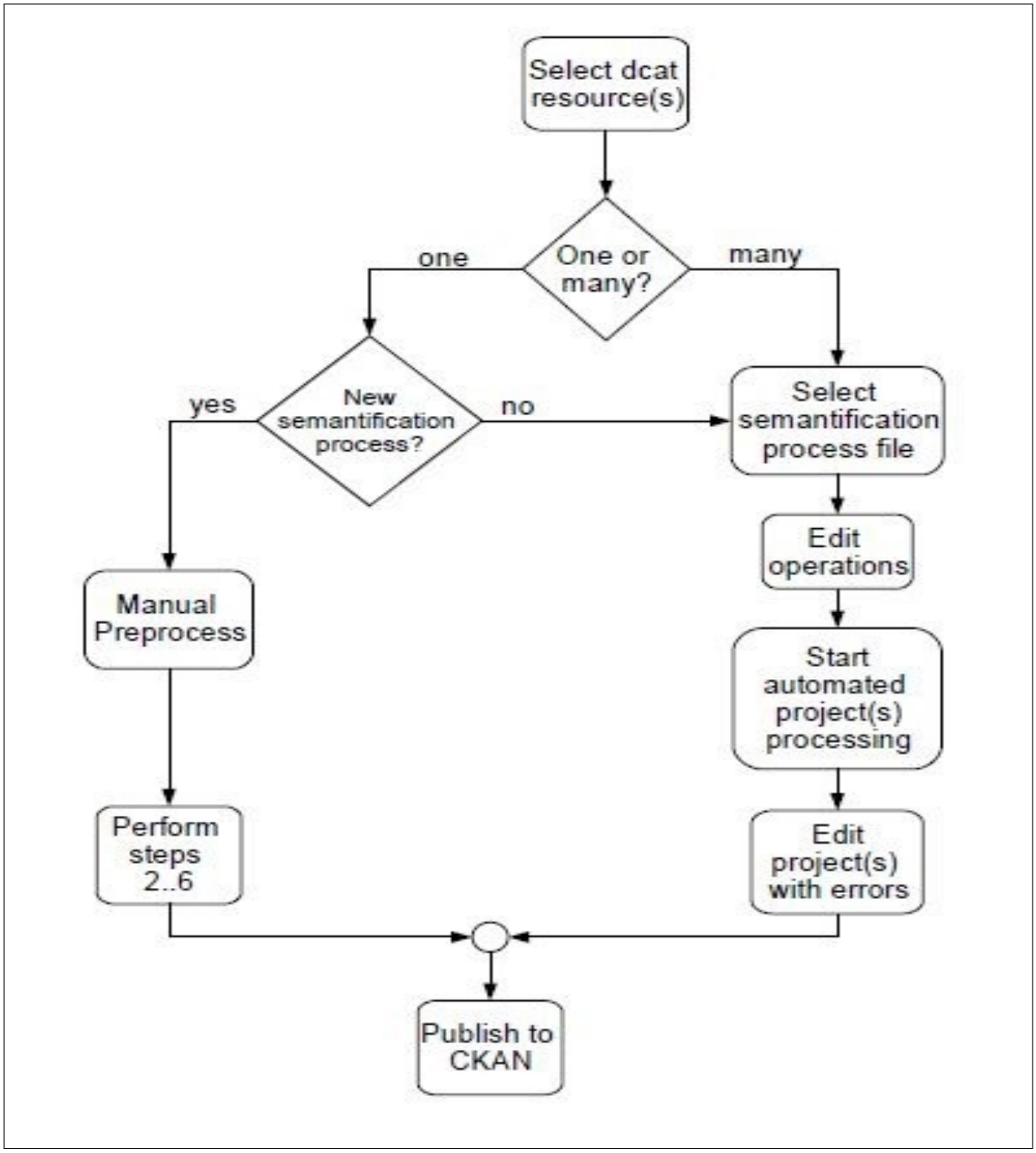


Figure 4.2: Workflow of dataset resources selection and editing

The pipeline has a high level workflow which uses preliminary steps and tool steps. This high level catalog workflow is used to select the resources and decide if the semantification will run in automatic or in manual mode. The high level semantification workflow is shown in Figure 4.2. The *manual preprocess* block corresponds to the preliminary steps and the *perform steps* block corresponds to the pipeline steps. In this section we present the high level semantification workflow. Prepocessing steps are introduces in next section and pipeline steps are given in Section 4.3.3. The pipeline can run in manual or automatic modes. The modalities of running the pipeline are discussed in Section 4.3.4.

The catalog importing workflow takes a catalog as input and checks for each resource in the catalog if the semantification process has been done before or if it is the first time to semantify the resource.

- If this is the first time to semantify the resource, then we execute both the preliminary and the pipeline steps manually. After the execution is done, a semantification process file is created. This file stores information about the decisions made while performing the semantification process.
- If this is not the first time to semantify the resource, then we use the semantification process file from a previous run to automate the processing.

4.3.2 The preliminary steps

Before using the tool steps of the semantifying pipeline, some data and services should be prepared to provide the required functions to the tool steps of the semantifying pipeline. This section lists the steps that install and prepare the data and services. We call these steps the preliminary steps because they are executed before processing the data with the tool

steps. The preliminary steps are done by experts who understand both the entity-centric model and the domain of the data. These steps are:

1. Installing the Peer Knowledge Core (PKC). The PKC is a software platform that has two parts: The Knowledge Base (KB) for storing knowledge elements such as words and concepts, and the Entity Base (EB) for storing entity elements such as entity types and attribute values.

2. Import natural language and concepts from Universal Knowledge Core (UKC). The UKC is a universal data set that can be imported into a Peer to provide knowledge elements. Natural language elements in the UKC are disambiguated to provide better semantic services.

3. Import the Entity Types (etypes) from the UKC. Initial entity types are imported from a global Entity Base such as Entitypedia into the Peer. This allows the pipeline user to reuse the existing entity types and their defined attributes.

4. Insert new knowledge elements. The natural language which is imported in step 2 from UKC may not contain all the words or concepts needed to semantify the datasets of a catalog. Experts will be able to add new elements to the local Peer knowledge base after analyzing the domain of the catalog.

5. Redefine the etypes. The imported etypes from UKC in step 3 can be extended or changed according to the specific data which will be processed later by the tool steps. Experts will analyze the data and modify the etypes as needed.

6. Redefine the identifying sets. New identifying sets can be created for new or existing etypes. And existing identifying sets can be also modified.

7. Enrich the UKC. The new and updated knowledge elements, etypes and identifying sets in steps 4, 5 and 6 can be promoted to the KB of the universal knowledge core (UKC). The UKC managers will check the elements and approve them before they are inserted into the UKC. These steps are described in terms of our specific implementation, but they can be generalized to other similar systems. For instance, natural language words can be imported from WordNET[2] and entity types can be imported from Freebase[3] .

4.3.3 The tool steps

The tools steps are composed of eight steps. The tools steps start by a *select* of a dataset and then processes it by *attribute alignment* and *attribute value validation*. Then the data passes through two core steps: *attribute value disambiguation* and *entity alignment*. The attribute value disambiguation step connects the data with unique identifiers for disambiguation and the entity alignment step resolves the conflicts between matched entities with different attribute values. The output is exported and published as a clean and semantically enriched version of the original dataset. This output can be visualized to the pipeline users if needed. The tools steps of the pipeline are shown in Figure 4.3.

[2]http://wordnet.princeton.edu
[3]https://www.freebase.com/

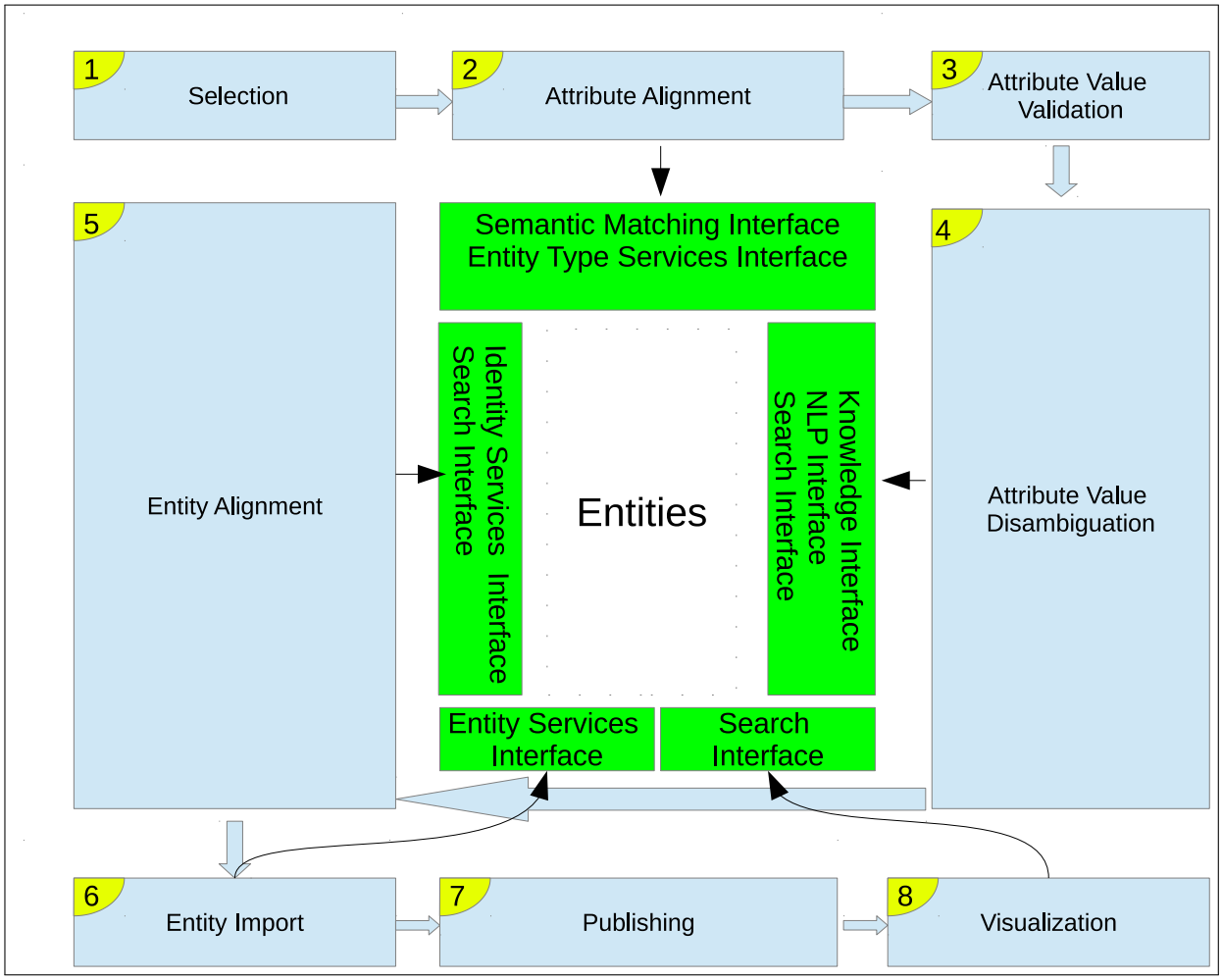


Figure 4.3: Overview of the import pipeline steps

Entities are at the center of the pipeline processing. They are stored in one or more entity bases. These entity bases are not part of the pipeline and the tool steps of the pipeline communicates with them through a programming interface. The programming interface calls the entity base services as presented in Section 4.1.1.

The eight tool steps are executed in sequence. Each step takes an input and produces an output. The input to the tool steps is an initial URL for a catalog that uses a DCAT vocabulary . The DCAT vocabulary allows us to browse the available datasets and access the actual resource of the

dataset. The resource can be for example a file to download or a SPARQL endpoint. The output of the tool steps is a semantically enriched version of the original dataset. The short description of the tool steps is given in Table 4.1.

N	Step Name	Description
1	Selection	Starting from an initial URL for a catalog that uses the DCAT vocabulary, we select a dataset from a list of datasets retrieved from the catalog and read it. We assume that the data is representable in a tabular form.
2	Attribute Alignment	We select a target type of entity, and then given the list of columns of the input dataset, we match them automatically with the list of attributes in the target type of entity. This will give the correspondence between the input columns and the output attributes. The user will then validate the schema matching result and add manually the possible missing correspondences and attributes.
3	Attribute Value Validation	This step applies format and structure validation and possible automatic transformations needed to have the input data in the expected format according to the selected entity type for the next steps of the pipeline. The correspondence between columns types and attributes of the entity type is used to automatically validate the column values given that the attribute definition includes the expected format, data type, and other possible conditions such as the allowed values (for example in the case of the gender attribute of person).

4	Attribute Value Disambiguation	This includes two operations: 1. Entity Disambiguation: used for finding the target entity in relational attributes. For example, if an entity *Trento* has a relational attribute *part-of* and the target of this relation is another entity *Italy*, then while we enrich the Trento entity we need to find the entity Italy and set it as the target entity of the relation. 2. Semantic Extraction of the free text: running the NLP pipeline on the free text attributes and extracting the concepts and entities from it. For example, if the gender attribute of a person entity is *Male*, then the extraction will provide the concept Male.

5	Entity Alignment	This step has three sub-steps: 1. Identity Disambiguation : Assuming that each row in the dataset represents an entity of the selected entity type in step 2, runs identity management algorithms to return a list of potential entities that match the input entity in the row and the knowledge base with the following potential types of results: (a) There is no match. (b) There is a match with exactly one entity. (c) There are several matching entities.

2. User validation or search: The user can accept or reject the output of the identity disambiguation step. In case of rejection the user can perform a manual search. The output of this step is one of the following:
   (a) An existing entity is chosen to be merged with the row.
   (b) A new ID is created for the row as a new entity
   (c) The user can skip the row and ignore it.
3. Create new ID or Merge :

In case of creating a new identifier, we use the Identity services interface to get the new ID.
In case of entity merging, we need to resolve possible conflict that can happen if the same attribute exists in two merged entities but with different values:

1. If the attribute value is single value at a time, then we may decide to override one of the values or, in presence of different time validities, keep both of them but with different provenance information.
2. If the attribute value accepts a list of values, then we need to check all the values in the two lists and merge the values.

Tasks that cannot be accomplished automatically in this step are resolved by the pipeline user or exported as crowdsourcing activities.

6	Exporting	The resulting semantified entities are stored in the knowledge base and optionally exported in some exchangeable format such as RDF.
7	Publishing	The final clean and semantically enriched entities are published back in the selected open data repository.
8	Visualization	The entities are visualized using a graphical user interface.

Table 4.1: Short description of the semantifying pipeline tool steps

There are five steps that need services from entity bases:

1. Attribute Alignment step needs a semantic matching interface, an entity type service interface.
2. Attribute Value Disambiguation step needs a knowledge interface, an NLP interface and search interface.
3. Entity Alignment step needs an identity service interface and a search interface.
4. Exporting step needs an entity service interface.
5. Visualization step needs a search interface.

The rest of this section gives more details for each step in the tool steps.

Selection

This step (see Figure 4.4) is the starting point of the tool steps. Its input is a catalog URL given as a string. A catalog is a collection of datasets and each dataset may contain one or more resources. A resource can be

any URL that gives access to data such as a spread sheet download link or a SPARQL end point. Selection allows browsing the catalog and choosing a dataset and a resource from it to be used as input to the pipeline. The output of this step is the selected resource, which we assume for the time being, to be in tabular format. Each row in the table can be considered to represent, at least partially, an entity.

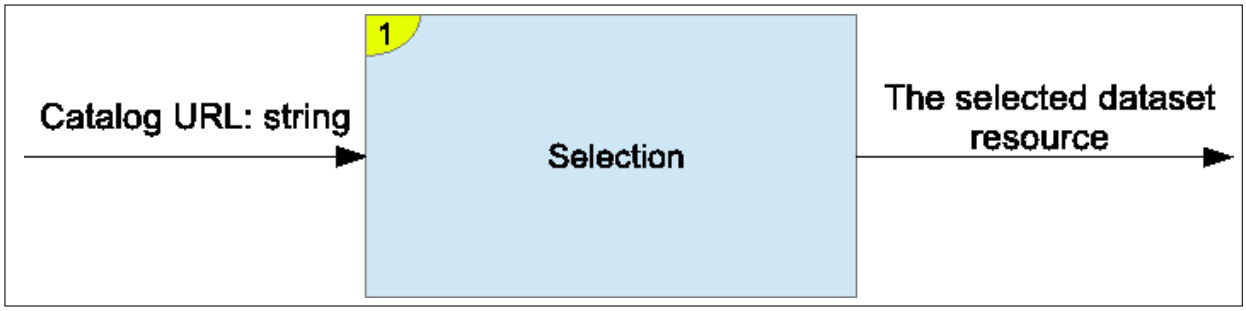


Figure 4.4: Step 1: Selection

The selected dataset resource is defined as a resource name, a list of column names and column values. All of these elements are strings. The BNF of the catalog, dataset and dataset resource are shown in Listing 4.2.

Listing 4.2: BNF for an open data catalog

```
Catalog ::= < {Dataset}* >
Dataset ::= < { DatasetResource }* >
DatasetResource ::= < ResourceName,
                      {ColumnName
                      {, ColumnValue}* }+ >
ResourceName ::= String
ColumnName ::= String
ColumnValue ::= String
```

Example: given the Catalog URL *http://dati.trentino.it* as input, the user navigates the Web interface of the catalog or uses the Web API to get

the following dataset resource: *http://www.meteotrentino.it/ws/service.asmx/listaCampiNeve*. The Resource Name is *Campi Neve* which is an Italian name for *Snow fields*. For the sake of having a complete running example, we assume that the column names and column values of this file are as given in Table 4.2. In this table, we have four columns and two rows. The column names are: *name*, *description*, *Position* and *Part-Of*. The first row shows a snow field with a name *Trento*. Its description is *a city in Trentino*, its position is *(4604, -11.07E)* and it is part of *Italy*. The second row shows another snow field with a name *Bolzano*. Its description is *a city in Alto Adige*, its position is *(4630, -11.21E)* and it is also part of *Italy*.

	Columns			
	Name	Description	Position	Part-Of
Rows	Trento	A city in Trentino	(4604, -11.07E)	Italy
	Bolzano	A city in Alto Adige	(4630, -11.21E)	Italy

Table 4.2: Columns and rows for an example dataset resource

Attribute Alignment

The attribute alignment step (see Figure 4.5) takes the selected dataset resource as input. It allows the user to choose the type of entity from a list of types. Reading the types of entities and their attributes from the knowledge base is done through a call to the knowledge base interface or it could be a call to another separate interface.

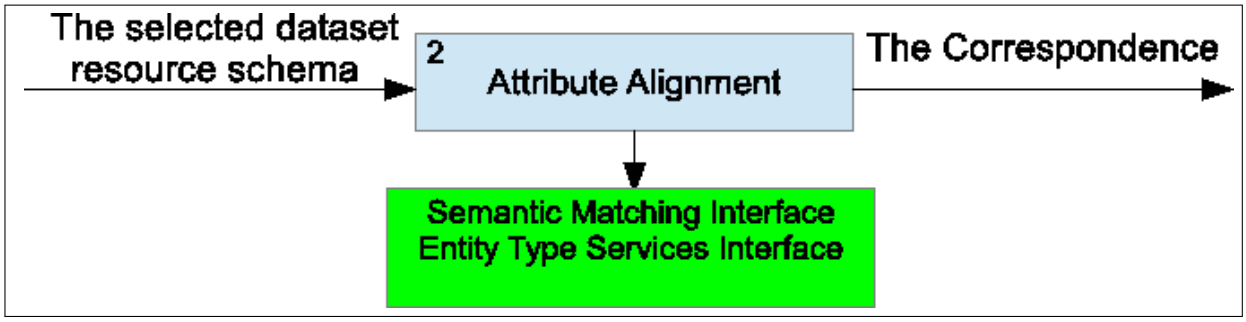


Figure 4.5: Step 2: Attribute Alignment

An entity type is automatically selected by running the schema matching operation between the selected dataset resource schema and each entity type that we read from the entity type service. The entity type which is semantically matched with the input resource schema with higher score is chosen. The user may also choose manually the type of entity from existing entity types.

The user can modify the entity type if needed. Manual modifications which are allowed in this step are adding new attribute definition and managing the list of attributes which uniquely identify the entity. These attributes which uniquely identifies the entity are called the unique indexes of the entity type. After choosing and modifying the type of entity, the pipeline uses the schema matching interface to match the columns of the input dataset with attributes of the chosen type of entity. The output of this step is the correspondence between the input columns and the attributes of the type of the entity.

The selected dataset resource schema is defined as a resource name and a list of column names. The BNF of the resource schema is shown in Listing 4.3.

Listing 4.3: BNF for Resource Schema

```
ResourceSchema ::= <ResourceName, {ColumnName}+ >
ResourceName ::= String
```

```
ColumnName ::= String
```

Where:

- ResourceName is the name of the dataset resource. It may correspond to the Name metadata of the dataset resource, or can be taken from the resource file name if the Name metadata is empty.
- ColumnName is the name of a column in the dataset resource. After reading the resource in tabular format, this usually corresponds to the table header.

The correspondence is defined as a list of correspondence items. Where a correspondence items is a source, relation and target. The Source is one column name; the relation is a semantic relation that can be: equivalence, more general, less general or disjointness; and the target is an attribute definition from an entity type chosen by the user. More details on semantic matching are given in [60]. The BNF of the correspondence is shown in Listing 4.4.

Listing 4.4: BNF for Correspondence

```
Correspondence ::= < {<Source, Relation, Target>}* >
Source ::= ColumnName
Relation ::= <EQUIVALENCE |
             MORE GENERAL |
             LESS GENERAL |
             DISJOINTNESS>
Target ::= AttributeDefinition
AttributeDefinition ::= <ID,
                         AttributeName,
                         NameSet,
                         AttributeType,
```

{MetaAttribute}+ >

Where:

- Source is a column name as defined in Listing 4.3.
- Relation is one of the following values: equivalence, more general, less general or disjointness.
- Target is an attribute definition from an entity type.
- ID is an internally managed unique identifier for attribute definition.
- AttributeName is a natural language independent concept that corresponds to the attribute name.
- NameSet is a set of names for the attribute definition. The first one is the default name.
- AttributeType is one of the following values: FreeText, Reference, Descriptive, QualitativeQuantitative or Relational.
- MetaAttribute is an attribute of the attribute definition.

The result of schema matching can be:

(a) There is no correspondence between the column and any attribute. In this case, the user can create a new attribute definition for the column or he can ignore it. An ignored column can be split into two columns or merged with another column in the step of data validation. (see attribute validation step)

(b) There is one correspondence between the column and an attribute. In this case, the correspondence information is stored in the tool memory so that it can be used in next steps.

(c) There is more than one correspondence between the column and more than one attribute. In this case, the user must choose one correspondence to be used in next steps of the pipeline and other correspondences will be ignored.

Example: The selected dataset resource has Resource Name: *Snow Field* and the following Column Names: Name, Description, Position, Part of. The entity type service returns the following entity type with name: *Location* and attributes Name, Description, Part of ,Latitude, Longitude. The Semantic Matching interface is called to match the selected dataset and the entity type. The result of semantic matching[4] is the correspondence which is shown in Table 4.3. The relation *more specific* means that the source concept is more specific than the target concept. I.e. *Short Name* is more specific than *Name*.

Source	Relation	Target
Snow Field	more specific	Location
Name	equivalent	Name
Description	equivalent	Description
Position	-	-
Part of	equivalent	Part of
-	-	Longitude
-	-	Latitude

Table 4.3: Example correspondence given by Semantic Matching

The source column Position is not matched with any attribute. The pipeline user can add it by searching through the Knowledge interface for the correct concept of the word Position. By investigating the values of the Position column, the pipeline user notices that it should be split into the two entity type attributes: Longitude and Latitude. So instead of creating

[4]We assume structural preserving schema matching (SPSM).

a new attribute for the column Position, the user just leave it for the next step.

The correspondence is used to link the source column names with the corresponding target attribute definitions. This is useful in next steps:

- In attribute value validation step, we use this information to know the data type and the range for the values of the column. (see attribute validation step)

- In attribute value disambiguation step, we enrich the columns which correspond to attributes that takes values of natural language strings using an NLP pipeline and we enrich columns which correspond to attributes that takes relational values using entity disambiguation. (see attribute value disambiguation step)

- In the entity alignment step, the correspondence is used while creating an entity that will be passed to the identity service. It is also used while merging two entities. (see entity alignment step)

Attribute Value Validation

The attribute value validation step (see Figure 4.6) applies structural and format transformations on the input data to make it compatible with the definition of the attributes given by the type of entity and to automatically discover errors when possible. The data validation step starts by comparing the value of the source column value with the expected data structure and format given by the definition of the attribute in the entity type.

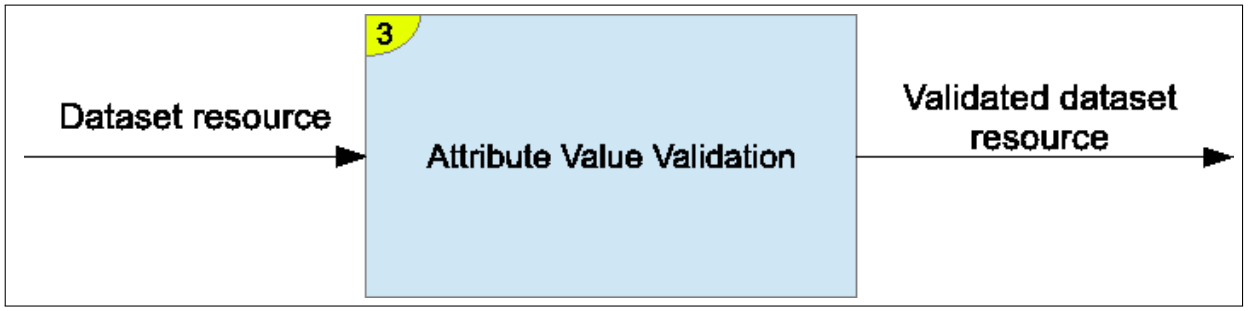


Figure 4.6: Step 3: Attribute Value Validation

The first validation is the Structure Validation: If there is a structure mismatch between the source column value and the expected entity type attribute value, then one of the following transformations could be applied:

1. Splitting one column into two or more columns corresponding to two or more attributes.

2. Merging two or more columns into one column corresponding to one attribute.

Example: Let us consider a structure validation example that is shown in Table 4.4. In the upper part (i), there is an input row which has a Name, Description, Position and Part-Of columns. All the columns are matched with their corresponding attribute definitions in the schema matching step except the Position column. The pipeline user notices that the Position column has the string value (4604, -11.07E) which represents the latitude and longitude of the location of the entity. The target entity type Location has two separate attributes Latitude and Longitude. The structure validation requires the splitting of the Location column values into the Latitude and Longitude attributes. The result of the splitting transformation is shown in the lower part (ii) of Table 4.4. Column names in this table with brackets around them are those which are matched with an entity type attribute.

	Attributes				
	[Name]	[Description]	**Position**		[Part-Of]
Input Row (Entity)	Trento	A city in North Italy	**(4604, -11.07E)**		Italy
	Attributes				
	[Name]	[Description]	**[Latitude]**	**[Longitude]**	[Part-Of]
Validated RowValidated Row (Entity)	Trento	A city in North Italy	**4604**	**-11.07E**	Italy

Table 4.4: An example for structure validation

The second validation is Format Validation with checks that are performed automatically if there is a one-to-one correspondence between the column and an attribute of a known type of entity:

1. Data type check: checking that the data type of the column is the same as the expected data type of the corresponding attribute in the entity type. For example, it checks that a numerical attribute does not contain string values.

2. Range check: checking the correct range values for attributes. For example, it checks that the postal code is in the acceptable range. The range restrictions come from the attribute definitions of the entity type as defined in [R4].

Example: Let us consider an input row shown in first row of Table 4.5. It has a Name, Description, Longitude, Latitude and Part-Of columns. The attributes Longitude and Latitude are defined to be of type Float. There are range constrains as following: Latitude range is from 0 to 90 and Longitude range is from -180 to 180. The values of Latitude and Longitude in the input row do not respect these constrains. The value of Latitude is 4604 which is not in the range (0, 90) and the value of Longitude is -11.07E has a letter E which is not accepted in the data type Float. The pipeline user uses the pipeline tool to fix the values of the two columns. The validated row is shown in the second row of Table 4.5, where value 4604 is corrected to 46.04 and the value -11.07E is corrected to 11.07.

	Attributes				
	[Name]	[Description]	[Latitude]	[Longitude]	[Part-Of]
Input Row (Entity)	Trento	A city in North Italy	**4604**	**-11.07E**	Italy
Validated Row (Entity)	Trento	A city in North Italy	**46.04**	**11.071**	Italy

Table 4.5: An example for format validation

After the validation is done, the output of the step becomes the validated version of the input dataset. Note that only those rows with valid formats will be part of the output of the step. Rows with invalid data will be filtered out.

Attribute Value Disambiguation

The Attribute Value Disambiguation step (see Figure 4.7) is a core step in the pipeline that takes a validated dataset and enriches it. It is composed of two sub steps: the entity disambiguation and the Natural Language Processing (NLP).

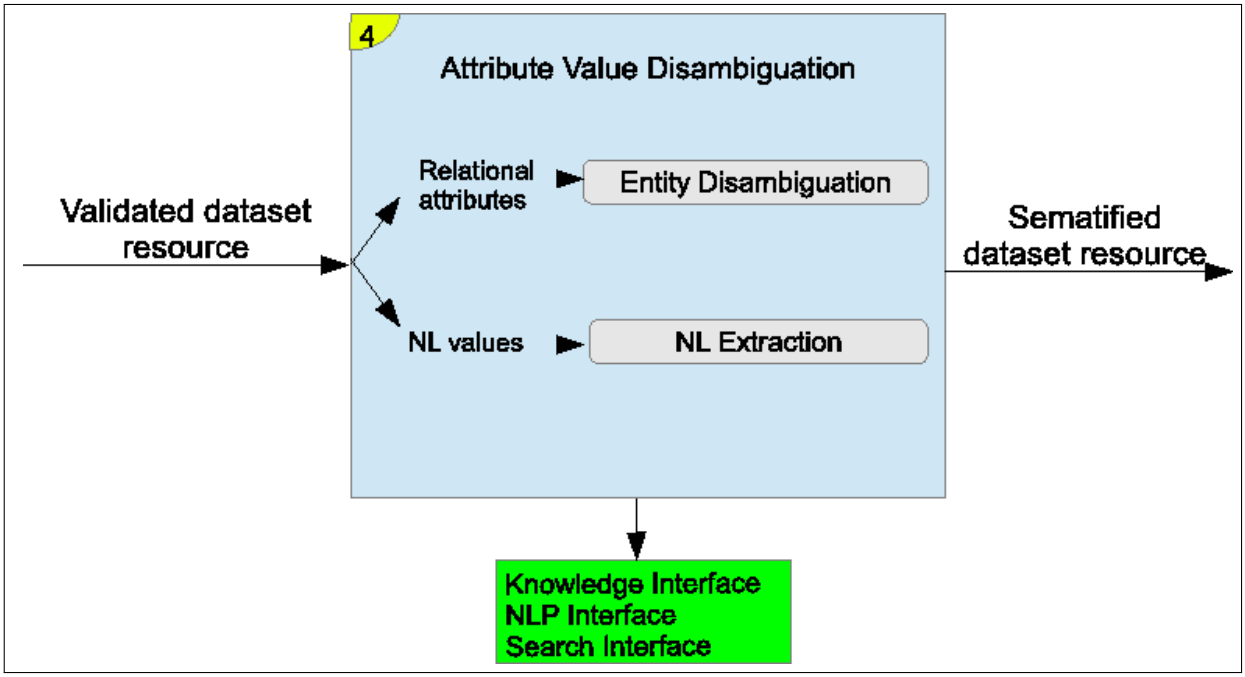


Figure 4.7: Step 4: Attribute Value Disambiguation

1. Entity disambiguation takes as input the entity names and/or any other attribute that can be used to compute the relational attribute value. The entity names come from columns that were identified with correspondences with relational attribute values of the selected types of entities. The entity disambiguation sub step assigns to each of the entity names the correct unique identifiers of the referred entity for disambiguation.

2. NLP takes as input the values of natural text columns in the validated dataset and extracts the concepts and entities mentioned in that natural language text.

The semantic enrichment step uses NLP interface for the disambiguation of concepts and entities. The tools will also call the knowledge service interface as needed.

Examples:

1. Let us consider the input row which is shown in the first row of Table 4.6. It has the attributes Name, Description, Latitude, Longitude and Part-Of. The Part-Of attribute is a relational attribute that gives the geographic entity which contains the Trento city. The value of this attribute is the string value *Italy*. After the entity disambiguation step, the identifier of the Italy entity will replace the string value *Italy*. We refer to the identifier of the Italy entity as [Italy]. The enriched row is shown in the second row of Table 4.6.

	Attributes				
	[Name]	[Description]	[Latitude]	[Longitude]	[Part-Of]
Input Row (Entity)	Trento	A city in North Italy	46.04	11.07	**Italy**
Enriched Row (Entity)	Trento	A city in North Italy	46.04	11.07	**[Italy]**

Table 4.6: Example for Entity Disambiguation

2. Let us consider the input row which is shown in the first row of Table 4.7. It has also the attributes Name, Description, Latitude, Longitude and Part-Of. The Description attribute takes values of natural language strings. The string value *A city in North Italy* is processed by NLP pipeline. The result is shown in the second row of Table 4.7 where the concept *city* and the entity *North Italy* are found and disambiguated. They are connected with their unique identifiers which we represent as [city] and [North Italy] .

	Attributes				
	[Name]	[Description]	[Latitude]	[Longitude]	[Part-Of]
Input Row (Entity)	Trento	**A city in North Italy**	46.04	11.07	[Italy]
Enriched Row (Entity)	Trento	**A [city] in [North Italy]**	46.04	11.07	[Italy]

Table 4.7: Example for NLP of natural language attributes

Entity Alignment

Entity Alignment (see Figure 4.8) is the second core step in the pipeline. It compares each row in the dataset resource with entities from an existing entity base. The goal is to find the identifier of the row or create a new one if needed. The Entity Alignment step takes the semantified dataset as input and produces a list of rows which represents entities to be created with their new identifiers, or merged with an entity with an existing identifier.

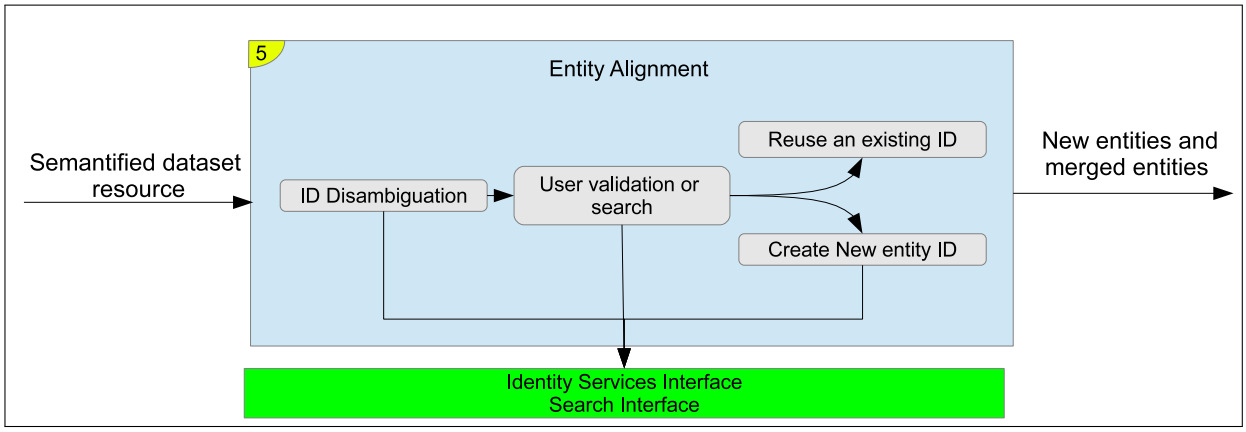


Figure 4.8: Step 5: Entity Alignment

Changes made to entities and attributes in this step are internal in the pipeline and will not affect the entity base. The changes will be reflected in the entity base later in the Exporting step.

There are three sub steps in the reconciliation:

Step 1: Identity Disambiguation Applies identity disambiguation on each row in the semantified dataset against entities that exist in a knowledge base. This is done by calling the Identity Service to perform identity disambiguation. The result of the identity disambiguation can be: no match, one match or more than one match. Mote details about identity management can be found in [82].

Step 2: User validation or search in which the user can accept or reject the output of the identity disambiguation step. In case of rejection the user can perform a manual search. Based on the output of the identity disambiguation step, we have the cases shown in Table 4.8.

Case	Result
1. If there is no match, then the user can choose to :	
i) Create an ID for the row as a new entity.	New ID
ii) Search manually for a match and reuse its ID.	Reuse ID
iii) Ignore the row	Ignore
2. If there is a match with one entity, then the user can choose to:	
i) Accept the matched entity and reuse its ID.	Reuse ID
ii) Reject the matched entity, search manually for a match and reuse its ID.	Reuse ID
iii) Reject the matched entity and create an ID for the row as a new entity.	New ID
iv) Ignore the row.	Ignore
3. If there is a match with more than one entity, then the user can choose to :	
i) Choose one entity and reuse its ID.	Reuse ID
ii) Reject the matched entities, search manually for a match and reuse its ID.	Reuse ID
iii) Reject the matched entities and create an ID for the row as a new entity.	New ID
iv) Ignore the row.	Ignore

Table 4.8: Cases of identity disambiguation

The output of this User Validation or Search step is one of the following:

(a) An existing entity ID is reused for the row,

(b) A new ID is created for the row as a new entity,

(c) The user can skip the row and ignore it.

Skipping the row is useful when the row is not actually an entity, so we cannot create an ID for it and we cannot reuse an existing ID and merge the row. In this case we do not store this row or any provenance information related to it.

Step 3: Create new ID or Reusing an existing ID

1. If the user chooses to create a new ID for the row, then the row is considered to represent an entity and a new ID is generated through the Identity Services interface.

2. If the user chooses to reuse an existing ID then we need to merger the row with the existing entity. We check the cell values of the input row against the entity attributes to see if there is any conflict. If one attribute has two different values in the two entities, then we perform conflict resolution for this attribute.

- If the attribute value is single value at a time, then we may decide to override one of the values or, in presence of different time validities, keep both of them but with different provenance information.

- If the attribute value accepts a list of values, then we need to check all the values in the two lists and merge the values, considering their provenance.

Tasks that cannot be accomplished automatically in this step are resolved by the pipeline user or exported as crowdsourcing activities.

The output of the reconciliation step is a list of rows with new IDs and a list of rows which reuses existing IDs. Rows with new IDs will be created as new entities and rows that reuse existing IDs will be merged with existing entities. New IDs given by the identity service and merged entities are stored only in the pipeline at this step. The changes will be exported in the entity base in the following step.

Examples:

1. Let us consider the input row in the first row of Table 4.9. This row represents an entity which we can pass to the Identity Service to see if there is an existing identifier for it. The identity service returned two entities that are possible matches for the input entity. They are shown in the last two rows of Table 4.9. The user can choose to merge the matched entity number 1 with the input entity because they represent the same real world entity of the Trento city even though some attribute values are different.

	Attributes				
	[Name]	[Description]	[Latitude]	[Longitude]	[Part-Of]
Semantified Row (Entity)	Trento	A [city] in [North Italy]	46.04	11.07	[Italy]

Matched Entity 1	Trent	A [city] in [North Italy]	46.04	11.071	[Trentino-Alto Adige]
Matched Entity 2	Trento	A [municipality] in [Philippines]	3.24	51.41	[Agusan del Sur]

Table 4.9: An example for ID Disambiguation

2. Let us assume that the user chooses to merge the matched entity number 1 with the input entity from the previous example. They are shown again in the first two rows of Table 4.10. The attribute values are merged based on the user choice. He chooses the Name, Description and Latitude attribute values from the first (input) entity, and he chooses the Longitude and Part-Of attribute values from the second (matched) entity. The merged entity is shown in last row of Table 4.10.

	Attributes				
	[Name]	[Description]	[Latitude]	[Longitude]	[Part-Of]
Semantified Row (Entity)	Trento	A [city] in [North Italy]	46.04	11.07	[Italy]
Matched Entity	Trent	A [city] in [North Italy]	46.04	11.071	[Trentino-Alto Adige]

Merged Entity	Trento	A [city] in [North Italy]	46.04	**11.071**	**[Trentino-Alto Adige]**

Table 4.10: An example for conflict resolution while merging two entities

For more information about merging entities and conflict resolution with provenance and authority, see Chapter 5.

Exporting, Publishing and Visualization

The exporting step takes as input all the semantified dataset already reconciled with their IDs disambiguated and stores them in the knowledge base or export them in some exchangeable format such as RDF. Provenance information can be exported also. See Chapter 3 and Chapter 5 for more information about provenance. The publishing step takes an input of data exchangeable format (e.g., CSV or RDF) and publishes it in a Web catalog, like a CKAN instance. The visualization is done by the web-based

user interface of Entitypedia. Exporting, Publishing and Visualization are shown in Figure 4.9.

Figure 4.9: Exporting, Publishing and Visualization steps

4.3.4 Running Modalities

The tool steps of the pipeline can run in two modalities: the manual and the automatic modes. This section describes them and gives the possible combinations of them.

Manual (interactive) Mode

The manual or interactive mode requires the pipeline user to perform the operations manually. The interactions between the user and the pipeline can be Manual Selection or Manual Processing. In the Manual Selection mode, the user selects the DCAT resources from the catalog manually. In the Manual Processing mode, the user performs the pipeline steps interactively.

Automatic (batch) Mode

The Automatic or batch mode does not require the pipeline user to perform any interactions with the system. If the system encounters an error or an ambiguity in the automatic mode, then it is reported in the log report. Similar to the manual mode, there are Automatic Selection and Automatic Processing. In the Automatic Selection mode, the dcat resources are selected automatically. In the Automatic Processing mode, the pipeline steps

are executed automatically without user interactions. Automatic processing cannot be done on new semantification processes

Combining modes

The manual and automatic modes can be combined in different import scenarios as shown in Table 4.11.

Selection Mode	Processing Mode	Description of the combined mode
Manual	Manual	The resource is selected manually and it is also processed manually. This is the case of the first time to run the pipeline on the resource.
Manual	Automatic	The resource is selected manually, and then the processing is done automatically. This can be done if the resource (or another version of it) is already processed by the pipeline before.
Automatic	Automatic	The resource is selected automatically and it is also processed automatically. This can be done if the resource (or another version of it) is already processed by the pipeline before.

Table 4.11: Combining the pipeline running modes

4.4 Summary

An entity base is a database which is designed to eliminate the lack of explicit semantics and the semantic heterogeneity problems. Before inserting a dataset in the entity base, entities must be imported from this dataset using a semantifying pipeline. The details of the pipeline steps are introduced with examples for each step.

This solution is implemented in OpenDataRise[5] which is an open source tool to clean and semantify datasets based on this proposed pipeline. This

[5]https://github.com/opendatatrentino/OpenDataRise

tool is currently used to import entities from open government data into an entity base[49].

Although this pipeline can be considered limited because its target database is only a database that stores entities and it can not be used with any generic database, we think that the entity base model is generic enough to handle most of the semantic web application scenarios. We tested the pipeline with a government scenario but it can be easily used in various personal and business scenarios.

Chapter 5

Entity Base Provenance

Recently an increasing number of open data repositories appear on the Web. A common category of the open data repositories is the CKAN catalogs. CKAN catalogs contain data that represents real world entities and their attributes. Entities can be imported from several catalogs to build web services; hence there is a need to trace the source of each entity and attribute value in a way that also handles the possible conflicts between attribute values coming from overlapping sources. We present here an approach for extending an incremental import process with a source tracing module that supports a conflict avoidance strategy.

5.1 Motivation

Open data repositories are software systems that provide packaging and distribution service for open data resources. One common example for these repositories is the CKAN repository system. It is used, for instance, in the Open Data Trentino web catalog[1]. These repositories contain data that represents real world entities and their attributes. To build web services that use this data, usually there is an import process that extracts entities and their attribute values from one or more CKAN catalogs into

[1] dati.trentino.it

a centralized entity base. If there is more than one source for an entity or an attribute value, the import process must be able to trace the source of each entity and attribute value which is created or updated during the import process.

One major problem with tracing sources is the tracing of data coming from heterogeneous data sources, where an object can be represented differently in each data source. For instance, importing data from two overlapped web sources into a single entity base that allows only one representation for an entity raises consistency issues due to possible conflicts. Moreover, resolving the conflicts manually becomes harder with the increasing size of data.

Figure 5.1 shows a running example which is not comprehensive but it gives an intuition into the problem. John, Bob and Maria are three persons who live in the three cities Trento, Mattarello and Rovereto respectively. The data about the city residents is stored by two organizations: Comune di Trento, for people in Trento; and Comune di Rovereto, for people in Rovereto. Mattarello is a small city located between the two cities and data of its residents could be stored in Comune di Trento, in Comune di Rovereto or in both.

John's data is stored in Comune di Trento, Maria's data is stored in Comune di Rovereto and Bob's data is stored in both. Bob could be represented in two different ways in each dataset, although the two representations refer to the same real-world person.

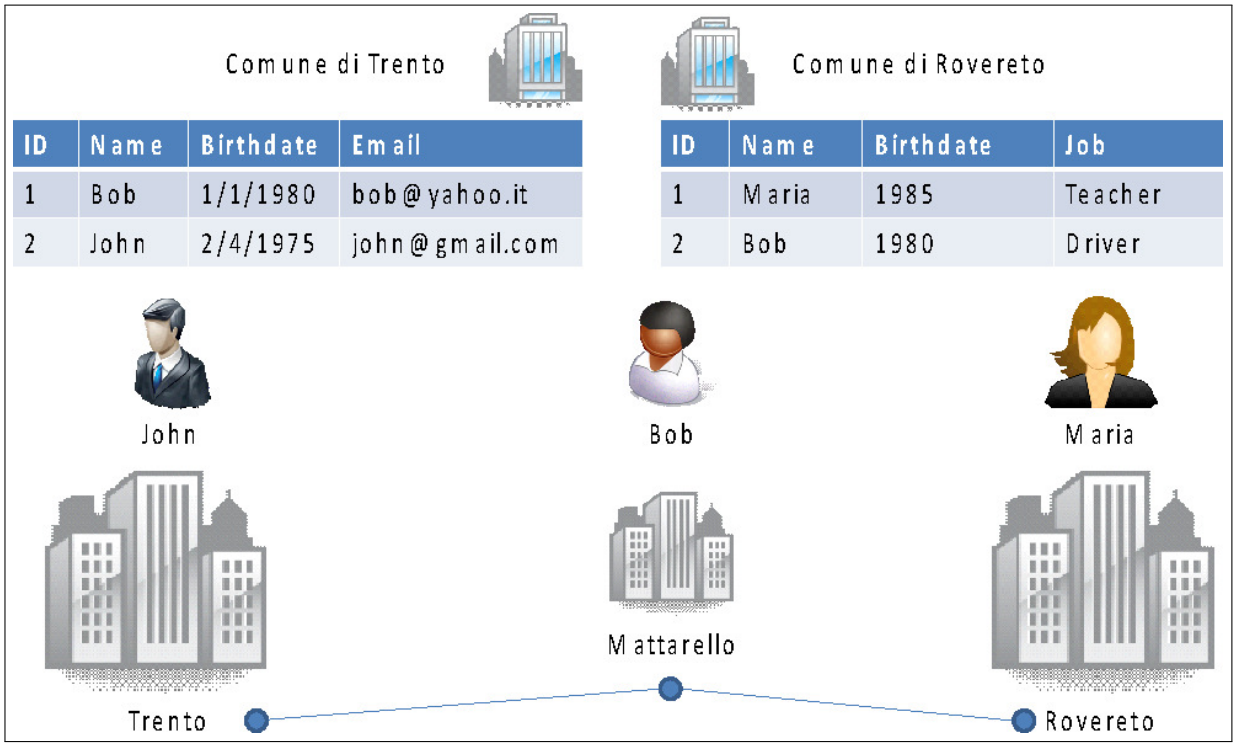


Figure 5.1: A motivation example: importing from two sources

To partially cope with this problem, we propose a source tracing module that extends an existing import process by making it tracing-aware. The source tracing module contains three tools: authority, provenance and evidence. Authority provides rules for overriding attribute values, provenance specifies the source of an attribute value and evidence links an entity with same external entities. An import process becomes tracing-aware by using the previously defined tools.

This problem has been studied before in the context of data fusion [52] and there are plenty of approaches to handle conflicts on the instance level. Our approach is to extend an incremental import process with a conflict avoidance strategy that gives some sources more trust (authority) than others. However, we go further than the state-of-the-art solutions by providing a source tracing solution that also handles conflicts between

attribute values in one integrated module; while focusing on the needs of open data catalogs.

The chapter is organized as follows. In Section 5.2, we describe the context and the problem of tracing sources with details of the DCAT catalogs, the entity base and the import process aspects; in Section 5.3, we present our approach to solve this problem by introducing the three tools: authority, provenance and evidence and we show an algorithm that use these tools to perform the import process with source tracing; in Section 5.4 we present the source tracing module; in Section 5.6, we apply our approach on a use case from the Open Data Trentino catalog; in Section 5.7, we give the conclusions and open problems for future work.

5.2 Problem

The problem of tracing sources is studied in the context of an import and export system. The general architecture of the system is shown in Figure 5.2. In this architecture, a peer contains a knowledge base (KB) and an entity base (EB). These two are populated by an import process which reads external datasets and writes the knowledge base elements in the KB and the entity base elements in the EB. We assume that the datasets are stored in a dataset management system such as CKAN. Datasets are always imported and exported from/to this dataset management system.

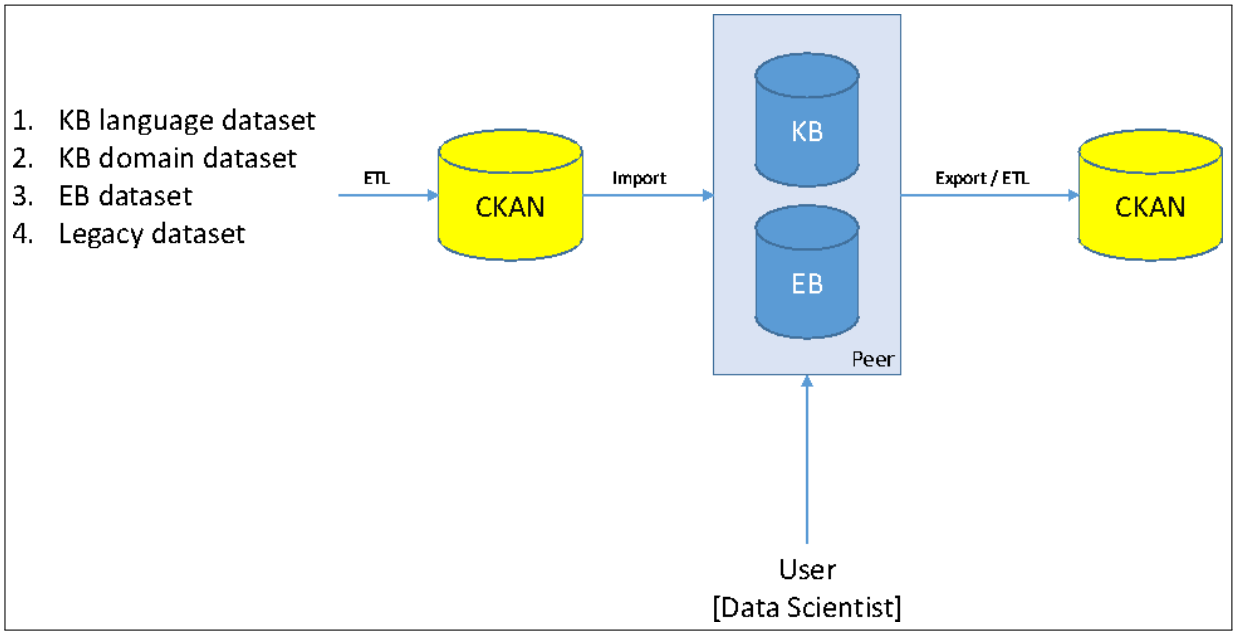


Figure 5.2: Overview of the import process scenario

The input datasets are classified into four types:

1. KB language dataset is a language dataset which is exported from a knowledge base. This dataset contains elements from Natural Language Core (NLC) and Concept Core (CC). These elements are shown in Figure 3.1.

2. KB domain dataset is a domain dataset which is exported from a knowledge base. This dataset contains elements from Domain Core (DC) and Etype Core (ETC). These elements are shown in Figure 3.1.

3. EB dataset is a dataset exported from an entity base. This dataset contains entities and attribute values.

4. Legacy dataset is any other dataset coming from external sources. This dataset may contain arbitrary knowledge base and entity base elements.

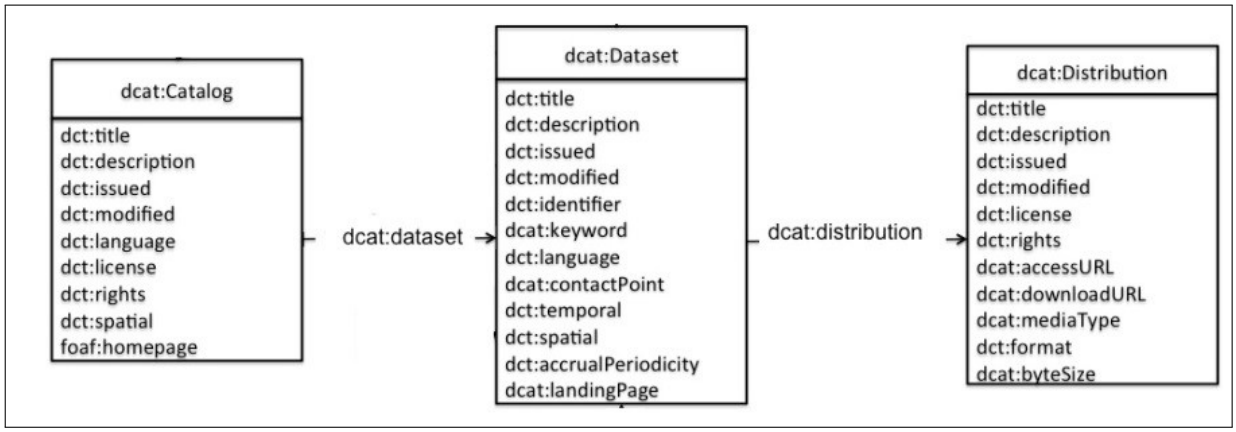


Figure 5.3: RDF Schema for Catalog, Dataset and Distribution in DCAT

The user who performs the import is a data scientist that must be qualified to perform the import process correctly. The import process takes as input a resource from a CKAN repository. This resource has its metadata described in DCAT vocabulary.

5.2.1 CKAN Repositories and DCAT vocabulary

DCAT[2] (Data Catalog Vocabulary) is an RDF vocabulary for describing datasets in a data catalog. A DCAT catalog can have one or more datasets, a dataset can have one or more distributions. The RDF schema for the catalog, the dataset and the distribution tables in DCAT is shown in Figure 5.3.

DCAT catalogs exist within a Web-based system called CKAN. CKAN (Comprehensive Knowledge Archive Network) is a dataset distribution system. Datasets are distributed as packages. Each package has one or more resource groups[3], and each resource group has one or more resources. The ER diagram for the package, the resource group and the resource tables in CKAN Version 2.2 is shown in Figure 5.4.

[2]http://www.w3.org/TR/vocab-dcat/

[3]Not to be confused with the groups table in CKAN schema that is used to group datasets together.

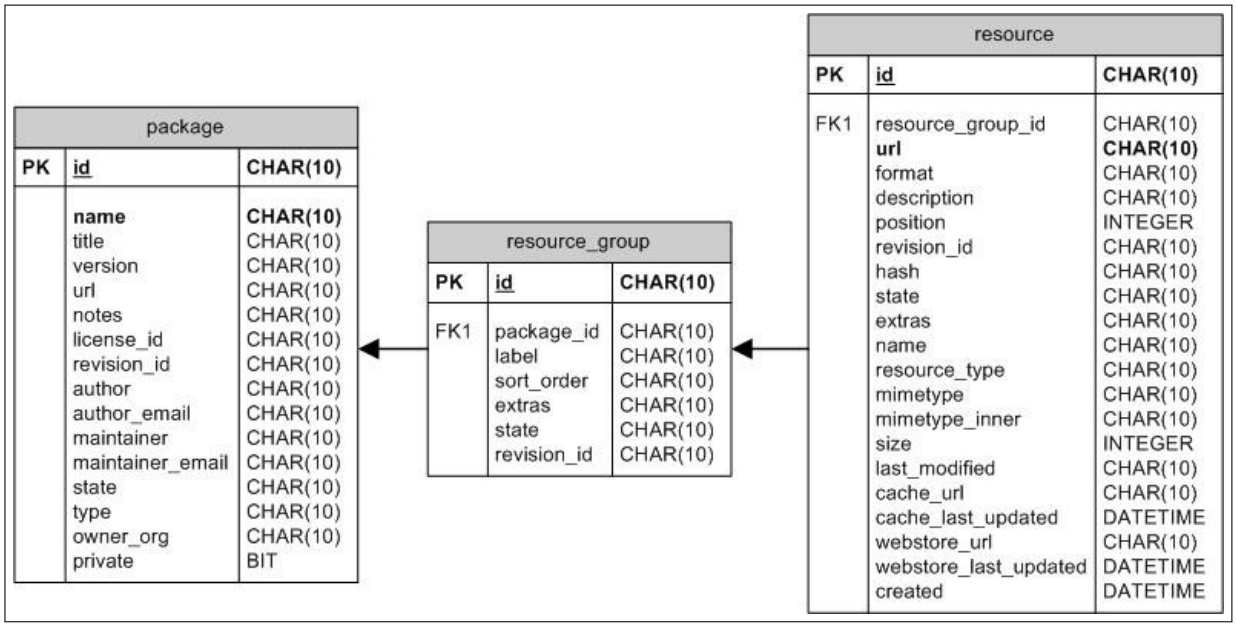


Figure 5.4: ER Diagram for Package, Resource Group and Resource in CKAN

Although CKAN and DCAT terminologies are different, we can find a correspondence between them. A CKAN installation corresponds to a DCAT catalog, a CKAN package corresponds to a DCAT dataset, and a CKAN resource corresponds to a DCAT distribution.

To use a unified terminology in this chapter, we assume that a DCAT catalog contains one or more datasets and a dataset contains resources. A resource is assumed to be a table of strings formatted in rows and columns. We also assume that each column has a name and that there is one identifying column for the table[4].

5.2.2 Import process aspects

The entity base is populated with entities through an import process. An import process can be, for instance, a generic work flow for importing any

[4]Resources which do not follow these assumptions must be converted to the assumed format before starting the import process.

dataset; a custom procedure for importing a specific dataset or a manual creation of entities and attribute values. We consider any import process that has the following three aspects:

- Partiality: The import process may take a partial input. This aspect applies to the list of resources in a catalog or to the list of columns and rows in a resource. For instance, a subset of resources in a catalog may be used as input to the import process instead of the whole catalog, or a subset of columns in a resource may be imported while ignoring the other columns.

- Overlap: Imported data may be disjoint or overlapped with existing entities and attribute values in the entity base. For instance, some entities which are extracted from a resource can be found also in the entity base. The overlapped entities may have different values for their attribute values.

- Multiple Imports: The import process may run multiple times on the same catalog. This aspect applies to the resources in the catalog and to the versions of each resource. For instance, when a new version of a resource is published in the catalog, the import process may run again to import the updated entities and attribute values.

5.3 Our approach

We propose a source tracing module that extends an existing import process by making it tracing-aware. An overview of our approach of extending the import process is shown in Figure 5.5. The source tracing module contains three tools: authority, provenance and evidence. An import process can access these tools using tracing-aware import procedures. This section

starts by introducing the authority, then it introduces the provenance and evidence; and finally it gives the tracing-aware importing procedures.

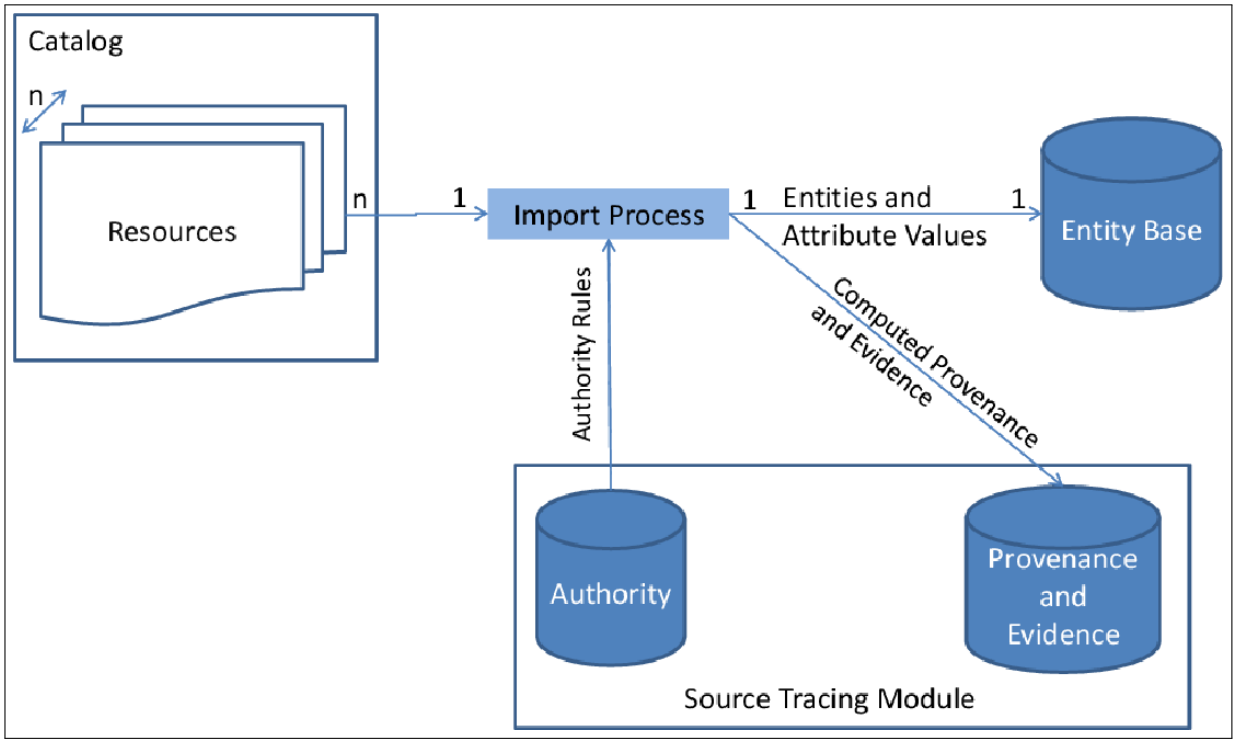


Figure 5.5: Extending the import process with source tracing module

5.3.1 Authority

The source tracing module in our approach has a source tracing tool called authority. Authority is a meta-attribute of an element in the entity base (i.e. an entity type, an attribute definition, an entity or an attribute value) that provides a connection between the element and the resource which has the authority to create or update it. Authority is specified through a set of authority rules. An authority rule is a relation between a resource and one or more elements which are called the scope, with a ranking value that is called the priority.

The scope specifies the set of elements that are affected by an author-

ity rule. We support four ordered levels of authority scope: (1) entity type, (2) a set of entities, (3) attribute definition and (4) attribute value. By default an authority on one level propagates to the next level, unless another authority is defined on the next level. The three aspects of the import process (partiality, overlap and multiple imports) can happen at any scope. The priority is a ranking value that is assigned to order if multiple sources are given authority for the same scope. This ranking is a total order. Authority scopes are shown in Figure 5.6.

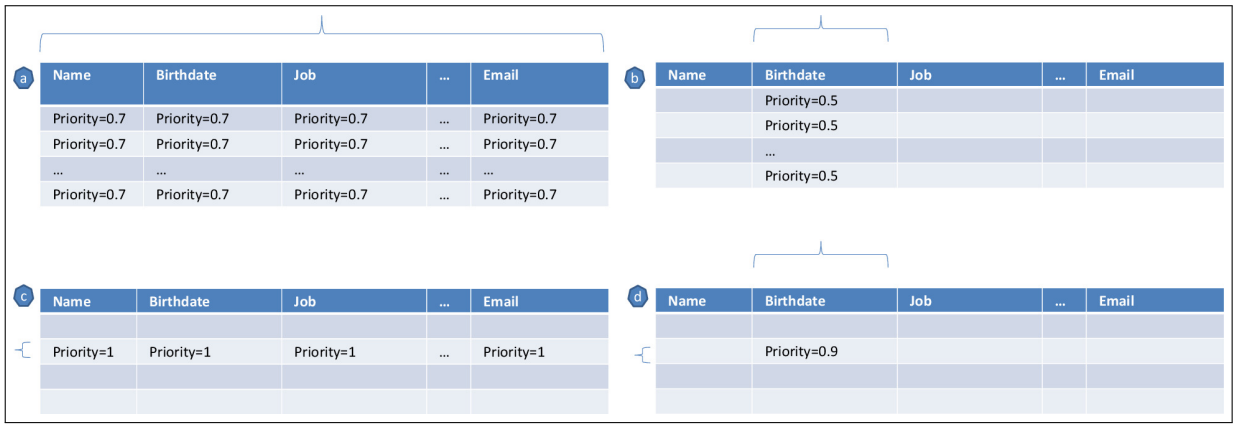


Figure 5.6: Authority Scopes. (a) Entity type (b) Attribute Definition (c) Entity Set (d) Attribute Value

Authority must be present always for each element that is going to be imported with the tracing-aware import process. Its purpose is to help in finding a winning resource if there is a conflict between two resources in an attribute value. Authority rules are defined in Listing 5.1:

Listing 5.1: Authority BNF

```
AuthorityRule ::= <     AuthorityRuleID ,
                        Resource ,
                        Scope ,
                        AuthorityPriority ,
                        [Note] >
```

```
Scope ::=          Etype |
                   EntitySet |
                   AttributeDefinition |
                   AttributeValue
AuthorityPriority ::= Float
EntitySet ::= {Entity}*
Note ::= String
Resource ::= URL
```

Where:

- AuthorityRule is a relation between a data source and a set of elements which are affected with this rule.
- Resource is the URL of the resource which holds the authority rights on the given scope.
- Scope is the set of elements that are affected by this authority rule.
- AuthorityPriority is a ranking value that is assigned to order if multiple sources are given the same scope.
- EntitySet is a set of entities. It can be a static list of entities or a query that returns a list of entities.
- Entity is presented in (todo: cite the chapter which introduces the entity base).
- Etype is presented in (todo: cite the chapter which introduces the entity base).
- AttributeDefinition is presented in (todo: cite the chapter which introduces the entity base).

- AttributeValue is presented in (todo: cite the chapter which introduces the entity base).

- Note is an optional string to record general notes.

For a set of authority rules associated with an entity base to be consistent, it must have the following properties:

1. There exists at least one authority rule that can be used to read or compute the priority value for every attribute value in the entity base.

2. A priority value that is defined on a scope is always greater than or equal to the priority value that is defined on a higher scope for the same resource.

Table 5.1 shows examples for authority rules. The first column has the Authority Rule and the second column shows the description of the rule . These examples are from the running example that we gave in Figure 5.1.

Authority Rule	Description
<1, Resource from Comune di Trento, Person (Etype), 0.7 >	The resource has authority on Person etype with priority value of 0.7
<2, Resource from Comune di Rovereto, Age (Attribute Definition), 0.5>	The resource has authority on Age Attribute Definition with priority value of 0.5.
<3, Resource from Comune di Trento, John (Entity), 1 >	The resource has authority on entities where birthplace is Trento with priority value of 1.
<4, Resource from Comune di Rovereto, Bob (Attribute Value), 0.9 >	The resource has authority on the value of the attribute Name for the entity Bob with priority value of 0.9.

Table 5.1: Examples for authority rules

5.3.2 Provenance and Evidence

In addition to the authority, which was presented in the previous subsection, the source tracing module that we propose has two other source tracing tools: provenance and evidence. Provenance is a meta-attribute that specifies the source of an attribute value. Evidence is an attribute of an entity that connects the entity with another external entity that represents the same real world object. It is similar to the same-as link in

OWL[5]. Provenance and evidence are defined using two helping elements: Reference and ImportProcess.

Reference is an element which gives the possibility to refer to external users and resources. Specific types of reference can provide a reference to the whole resource, to a part of the resource (such as a row in a tabular resource) or to a value in a row with a transformation applied on it. The metadata of the resource and the user are stored as attributes of an entity in the EB (See Section 5.3.2).

ImportProcess is an element that represents a process for importing entities and their attribute values process from an external resource into an entity base. ImportProcess supports the three aspects of an import process: partiality, overlap and multiple imports (See Section 5.2.2).

References

We define two major types of references: UserReference and ResourceReference. UserReference is a reference to a user that can be a human user or a software agent. The ResourceReference is a reference to a resource in an external dataset ResourceReference has two sub types: ResourcePartReference which is a reference to a specific part in the resource and ResourceValueReference which is a reference to a specific value in the resource with a transform that was applied on it during the import process.

Reference types and their examples are shown in 5.7. A UserReference simply refers to a user. A user is the person or the software agent which creates or modifies the element during the import process. A ResourceReference refers to the dataset resource. The example shown in the figure shows a dataset resource which is in tabular format. The first row in the table shows the names of the properties and the rows below are the values of these properties. A special property is the identifier property which is

[5]http://www.w3.org/TR/owl-ref/#sameAs-def

the External ID in this example. The value of the identifier property is unique for each row.

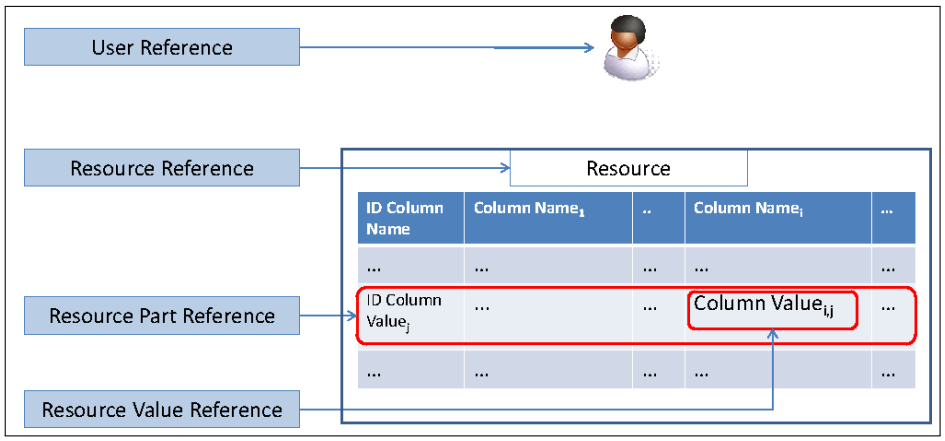


Figure 5.7: Reference types

A reference is data structure used to provide external references. There are four reference types:

1. User Reference (User). Refers to a user who participated in the import and his editorial role.
2. Resource Reference (Resource). Refers to an external resource.
3. Resource Part Reference (ResourcePart). Refers to a part of an external resource. (e.g. a row in a table)
4. Resource Value Reference (Value). Refers to a value in an external source and a transform that was applied on it.

Listing 5.2: Reference BNF for EB Provenance

```
Reference       ::=     User
                        | Resource
                        | ResourcePart
```

```
User              ::=   Id
                        , EditorialRole

EditorialRole     ::=   String

Resource          ::=   Id
                        , {ResourcePart}*

ResourcePart      ::=   Identifier
                        , URI

Value             ::=   ResourcePart ,
                        SourceProperty
                        [ , Transform ]

Identifier        ::=   Name
                        , Value

Name              ::=   String
Value             ::=   String
URI               ::=   String
Id                ::=   Long

Transform         ::= String

SourceProperty    ::=   < PropertyIdentifier ,
                        IsImported ,
                        [ , PropertyName ]
                        [ , TargetAttributeDefinition ] >
```

```
PropertyIdentifier              ::= String
IsImported                      ::= Yes | No
PropertyName                    ::= String
TargetAttributeDefinition       ::= Id
```

Where:

- Reference: is an external reference to a user, a resource or a resource part.
- User: is a reference to a user with the editorial role assigned to this user. The user is represented as an entity with a set of attributes defined in Section 5.3.2.
- Resource: is a reference to an external resource. The resource is represented as an entity with a set of attributes defined in Section 5.3.2.
- ResourcePart: is a reference to a part in an external resource.
- Identifier: is the identifier of the part in the external resource.
- Name: is the name of the identifier of the resource part.
- Value: is the value of the identifier of the resource part.
- URI: is the universal resource identifier of the resource part.
- Id: is the internal identifier of a resource or an entity in an entity base. More information on the resource and user entities are given in the Section 3.3.1.
- EditorialRole: is the editorial role that can be assigned to a user.

- Value: is a reference to an external value of a property associated with an element in an external dataset resource. It uses ResourcePart to specify the element which holds the property. DatasetValueReference stores the external property which is used to import the value and the transform that has been done on the value.
- Transform: is a string that codifies the transform which was applied on the external value during the import process.
- SourceProperty: is a field which represents a property in the dataset. It stores the identifier of the property and a flag saying if it is imported or not. It optionally stores a user friendly name of the property.
- PropertyIdentifier: is the name of the property as it is defined in the external resource.
- IsImported: is a flag that says if the property is imported in the import process or not.
- PropertyName: is a user friendly name for a property in the external resource.
- TargetAttributeDefinition: is the identifier of an attribute definition which is mapped to this source property. It can be empty is the source property is not mapped, but this source property can not be imported. It must have IsImported = false.

Modeling Users and Resources

6

We present here the minimal set of attributes for modeling user and resource entities. The general attributes of an Entity are given in Table

[6]this section is identical to 3.3.1 except that domain expertise is added to person and the etypes of DCAT resources are defined.

5.2. These attributes are inherited by both the user and resource entities. In this table (and also in the next tables) The first column shows the attribute, the second column shows the data type and the third column shows the reference dataset which gave the definition of the attribute. The <P >symbol beside a data type means that the corresponding attribute is permanent (i.e. it is not constrained with a time validity, but it is always valid).

Attribute	Data type	Reference Dataset
Name	NLString []	
Class	Concept <P >	
SURL	String <P >	UKC system itself

Table 5.2: Entity attributes

- 1. Person

- Category: personal

Attribute	**Data type**	**Reference Dataset**
Name	NLString []	Person herself
Class	Concept	UKC system itself
Gender	ENUM(MALE, FEMALE) <P>	person herself
birth Bate	Date <P>	Person herself
Email	String	Person herself

Relation	Entity type	Reference Dataset
Country of citizenship	Country	Person herself
Country where hiving now	Country	Person herself
City where living now	City	Person herself
Photo	String	Person herself

- Category: biography

Attribute	Data type	Reference Dataset
Degree	ENUM(PhD, MASTERS, BACHELOR, HIGH SCHOOL, NONE) [] <P>	Person herself (Note that with array symbol we mean that a person ctn select more than one degree)
Work	Boolean	Person herself
Student	Boolean	Person herself

- Category: user

Attribute	Data type	Reference Dataset
Login name	String	Person herself
Password	String	Person herself
Joining date	Date	Person herself
Leaving date	Date	Person herself

Relation	**Entity type**	**Reference Dataset**
Recommender	Person <P>	UKC existing user (possibly suggested by person herself)

- Category: expertise

Attribute	**Data type**	**Reference Dataset**
Language proficiency	<Language, Level> []	Person herself
Domain proficiency	<Domain, Level> []	Person herself
• Level	ENUM(A1, A2, B1, B2, C1, C2)	person herself

Relation	**Entity type**	**Reference dataset**
Language	Language	Person herself

- 2. Resource

- Category: general

Attribute	Data type	Reference Dataset
Name	NLString []	Entity importer herself
Class	Concept <P>	Entity importer herself
description	SString <P>	Entity importer herself
License	String	Entity importer herself
Note	NLString <P>	Entity importer herself
Version	String <P>	Entity importer herself
Release	String <P>	Entity importer herself
Date of publication	Date <P>	Entity importer herself

Relation	Entity type	Reference Dataset
Owner	Person and/or Organization	Entity importer herself

- Category: resource identity

Attribute	Data type	Reference Dataset
Homepage URL	String	Entity importer herself
Resource URL	String	Entity importer herself

Attribute	Structured type	Reference Dataset
KiDF URL	KiDF URL	Entity importer herself

- Structured types

KiDF URL

Category: recourse identity

Atbritute	Data type	Reference dataset
Name	NLString []	Entity importer herself
Description	SString	Entity importer herself
Note	NLString	Entity importer herself
URL	String	Entity importer herself

This section shows how to capture DCAT metadata using resource entity types. The DCAT catalog, dataset and distribution are mapped to three entity types Catalog, Dataset and Distribution. The three etypes are sub types from the Resource etype, hence they inherit all the attribute definitions which are already defined in this entity type. See Figure 5.8.

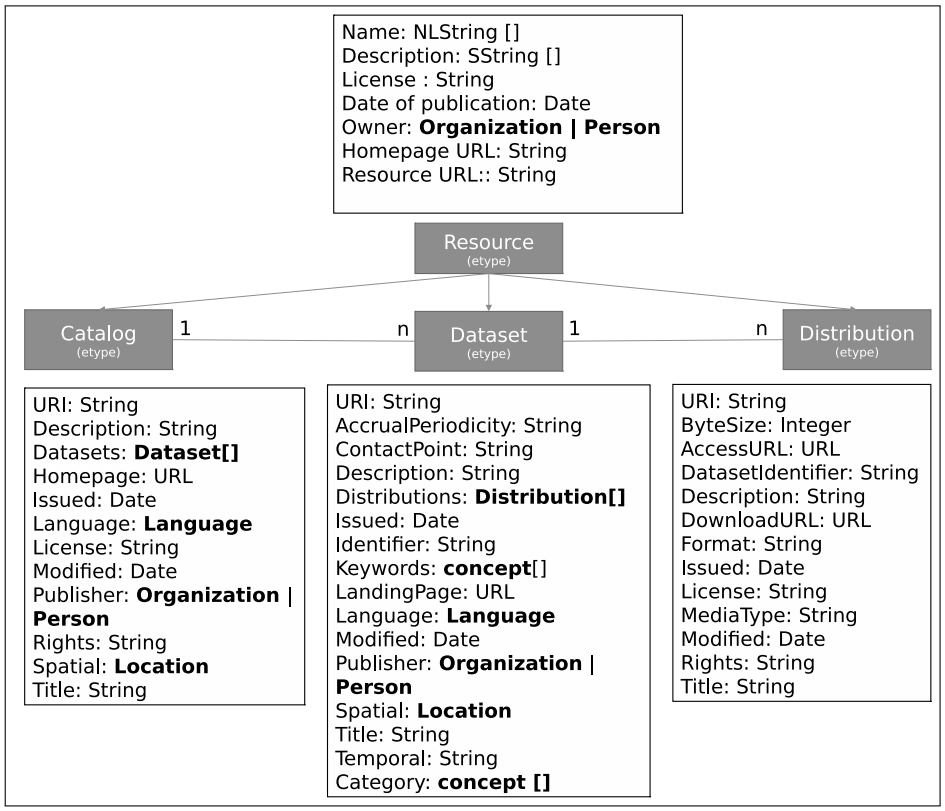


Figure 5.8: Catalog, Dataset and Distribution etypes

1. Catalog

Category: general

Attribute	Data type	Reference Dataset
URI	String <P>	Entity importer herself
Modified	Date	Entity importer herself
Rights	String	Entity importer herself

Relation	Entity type	Reference Dataset
Language	Language <P>(Defined in Person etype)	Entity importer herself
Datasets	Dataset[]	Entity importer herself
Spatial	Location	Entity importer herself

2. Dataset

Category: general

Attribute	Data type	Reference Dataset
URI	String <P>	Entity importer herself
AccrualPeriodicity	String	Entity importer herself
ContactPoint	String	Entity importer herself
Identifier	String <P>	Entity importer herself
Keywords	Concept[]	Entity importer herself
Modified	Date	Entity importer herself
Temporal	String	Entity importer herself
Theme	Concept[] <P>	Entity importer herself

Relation	Entity type	Reference Dataset
Language	Language <P>(Defined in Person etype)	Entity importer herself
Distributions	Distribution[]	Entity importer herself
Spatial	Location	Entity importer herself

3. Distribution

Category: general

Attribute	Data type	Reference Dataset
URI	String <P>	Entity importer herself
ByteSize	Integer	Entity importer herself
AccessURL	String <P>	Entity importer herself
DatasetIdentifier	String <P>	Entity importer herself
Format	String <P>	Entity importer herself
MediaType	String <P>	Entity importer herself
Modified	Date	Entity importer herself
Rights	String	Entity importer herself

The following table shows a mapping between the attributes of the parent etype Resource to the child etypes Catalog, Dataset and Distribution:

Resource Attribute	Catalog Attribute	Dataset Attribute	Distribution Attribute
Name	Title	Title	Title
Description	Description	Description	Description
License	License	-	License
Date of Publication	Issued	Issued	Issued
Owner	Publisher*	Publisher*	-
Homepage URL	Homepage	Landing page	Access URL
Resource URL	-	-	Download URL

*Assuming the publisher is the owner of the catalog or dataset. But this must be validated by the pipeline user when creating the resource entity.

ImportProcess element

In addition to the reference and the resource metadata elements, there is another helper element which is the import process. An import process (ImportProcess) is a representation of a process that imports entities and their attribute values from an external dataset resource, and creates or modifies the corresponding entities and attribute values in an entity base. The import process runs incrementally, i.e. it imports one resource at a time. Resources are considered to be in tabular format as shown in Listing

5.2. An import process can be partial, overlapped and it can be performed multiple times (See Section 5.2.2).

The partiality aspect requires us to store for each import process which columns has been imported. The multiple imports aspect requires us to store the timestamp of each import process.

Listing 5.3: ImportProcess BNF

```
ImportProcess      ::= < Resource ,
                         ModificationDate ,
                         User ,
                         {SourceProperty}* >
ModificationDate   ::= Date
```

Where:

- ImportProcess: is a representation of a process that leads to creation or modification of entity base elements (i.e. entities and their attribute values).
- Resource: is defined in Listing 5.2.
- ModificationDate: is a timestamp specifying the date of the import process. The import process runs incrementally, i.e. it imports one resource at a time.
- User: is a reference to the user who is responsible for the import process. It is defined in Listing 5.2.

Provenance and Evidence elements

An import process runs on an external resource and extracts entities and their attribute values from it. Before creating or updating the entities and their attribute values in the entity base, a tracing-aware import process

creates a graph of elements between the external source and the entity base. This graph is shown in Figure 5.9. The ultimate goal of this graph is to trace the sources of each element in the entity base. The graph is connected to the entity base through two source tracing tools: provenance and evidence.

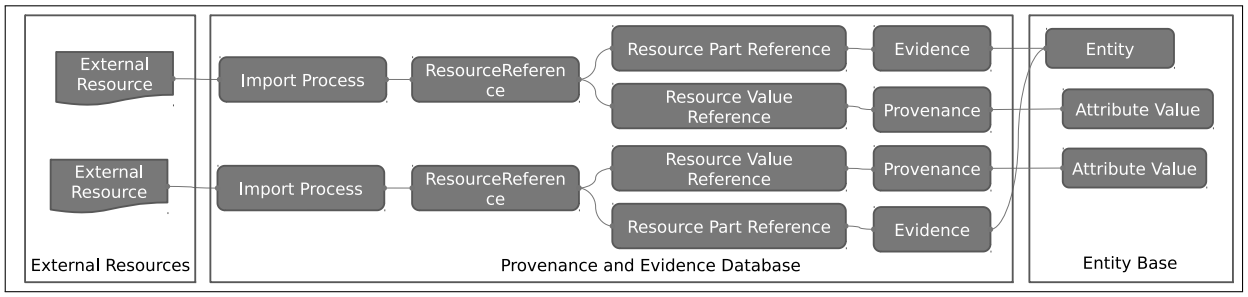


Figure 5.9: Provenance graph for the entity base

Provenance is a meta-attribute that specifies the source of an attribute value. Evidence is an attribute of an entity that connects the entity with another external entity that represents the same real world object. The BNF of provenance is given in Listing 5.4 and the BNF of the evidence is given in Listing 5.5.

Listing 5.4: EBProvenance BNF

```
EBProvenance        ::=  <AttributeValueID,
                           Accuracy,
                           ResourcePart,
                           ImportProcess
                           [,Note]>
ProvenanceID        ::= Long
AttributeValueID    ::= Long
Accuracy            ::= Float
Note ::= String
```

Where:

- EBProvenance: is a source tracing tool which is used to store the source of an attribute value.
- AttributeValueID: is the unique identifier for the entity base attribute value which is associated with the provenance. It is generated and maintained internally by the entity base.
- Accuracy: is a Float in the range [0, 1] providing the degree of confidence that the value is correct. The default value for accuracy is 0.9 for the data that is manually processed. Accuracy is used only for attribute value provenances.
- ResourcePart: is defined in Listing 5.2.
- Import Process: is defined in Listing 5.3.
- Note: is a text field to store user notes.

Listing 5.5: EBEvidence BNF

```
EBEvidence ::= <EntityID ,
                { ResourcePart }+ >
EntityID ::= SURI
```

Where:

- EBEvidence: is a source tracing tool for linking an entity with an external authority data set. It provides the linking between external sources thus it corresponds to the linking phase in linked data. Its role is similar to the 'same as' link in OWL with the addition that it also records according to whom the same as link is valid. It is also an indication that this piece of information is correct by providing the information source. Evidence is applied only to Entities.

- ResourcePart: is defined in Listing 5.2.
- EntityID: is the unique identifier of the entity generated and maintained by the identity management module in the entity base. SURI is defined in (todo: reference to the chapter the describes the entity base)

5.4 Source Tracing Module

The three source tracing tools which are presented so far are authority, provenance and evidence. These tools are stored in a database called the source tracing module. The source tracing module is a module that extends the import process by providing read/write operations for the source tracing tools.

The services which can be performed by the source tracing module are:

- AddEvidence: adds an external reference as evidence to the given entity.
- SetProvenance: sets the provenance for the given attribute value.
- ReadAttributeValueProvenance: reads the provenance of the given attribute value.
- ReadResourcePriority: reads a float number that represents the value of the priority which a resource has on a specific attribute value.

An import process can be extended to a trace-aware import process by updating its procedures to use the source tracing module. Any import process that creates entities and attribute values has a procedure that is used to call the entity base to do entity import and attribute value import. To extend the import process, we change the procedure that performs

entity import and the procedure that performs attribute value import with the trace-aware importing procedures which are shown in Listing 5.6.

The trace-aware importing procedures are:

- ImportEntity: is a method that creates an entity if it does not exist. It adds evidence on the new entity or on an existing entity.

- ImportAttributeValue: is a method that creates an attribute value if it is new or if the attribute definition if multi-valued; and it updates an existing single-valued attribute value if the authority rules allow. Authority rules allow updating an attribute value if the resource which is imported in this import process has a higher authority on the attribute value than the resource from which the current value was imported. The method sets also the provenance of the attribute value if it is new or updated.

- GetPriority: is a helper method to perform reading of the float number that represents the value of the priority which a resource has on a specific attribute value. If there is no priority value is set on this level, it checks the level below.

Listing 5.6: Tracing-aware importing procedures

```
function ImportEntity( Entity entity,
                       ResourcePartReference reference) {
        If (GetEntityIdentifer(entity) == null) {
                CreateEntity(entity)
        }
        AddEvidence(entity, reference);
}

function ImportAttributeValue( AttributeDefinition ad,
                               AttributeValue newValue,
                               Resource resource,
                               ResourceValueReference reference) {
        currentValue = ReadAttributeValue(ad);
        If(currentValue == null OR IsMultiValuedAttribue(ad)){
                CreateAttributeValue(ad, newValue);
                SetProvenance(newValue, reference);
        }

        Else if (currentValue <> newValue) {
         ExistingProv = ReadAttributeValueProvenance(currentValue);

         If(GetPriority(resource, currentValue, 4) >
            GetPriority(ExistingProv.resource, currentValue, 4))
                {
                 UpdateAttributeValue(ad, newValue);
                 SetProvenance(newValue, reference);
                }
         }
        }
}

function GetPriority(Resource resource,
                     AttributeValue attributeValue,
                     Level level) {
        priority = readResourcePriority(resource, attributeValue, level)

        If(priority != null)
             Return priority;
        Else
             Return GetPriority(resource, attributeValue, level -1);
        }
}
```

The strategy that we used in this source tracing module is to avoid conflicted attribute values by giving some resources more authority than others. As we said before in Chapter 2, there are two other know classes of strategies in the literature: conflict ignoring and conflict resolution. In conflict ignoring strategies, two attribute values which are in conflict may be created in the entity base. This strategy is not acceptable for the entity base because it leads to inconsistency.

Since we consider an incremental import process, we have a maximum of two values that can be in conflict: the currently imported value and the existing value. This makes conflict resolution strategies such as taking the average or the most occurring value not appropriate. However we may use the strategy of conflict resolution by taking the most recent value if the two resources are on the same level of authority.

5.5 User Interface

This section shows initial mock-ups for the user interface (UI) that uses the source tracing module. It is divided into two parts: (1) UI for the authority management and (2) UI for provenance and evidence visualization.

5.5.1 Authority management

There are two interfaces for authority management: one for the expert user and one for the end user. The expert user will be using the interface to add or delete authority rules. The end user will be using the interface to query authority rules. Expert UI is shown in Figure 5.10.

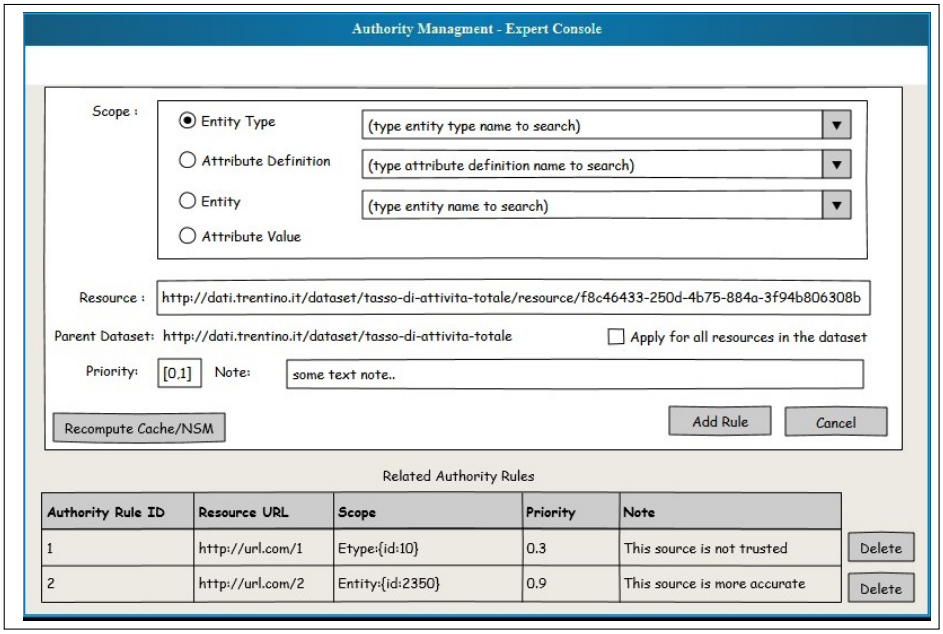


Figure 5.10: Authority Management Expert Console UI

Where:

- *Scope* is a box of four options:

  1. *Entity type* if this option is selected, then the other two search boxes are disabled. The user can choose the etype by typing its name and search for it in etype search box.
  2. *Attribute Definition* if this option is selected, then the entity and attribute value search boxes are disabled. The user first chooses the etype, and then he will find the attribute definitions of that etype in the search box. He can also type a name to start searching the list.
  3. *Entity* if this option is selected, then the attribute definition and the attribute value search boxes are disabled. The user first chooses the etype, and then searches for the entity name in the entity search box.

4. *Attribute Value* if this option is selected, then all the three search boxes are enabled. The user first chooses the etype, and then he can choose the attribute definition and the entity which identify together the specific attribute value.

- Resource text input: is the input box for the resource URL
- Parent Dataset label: is automatically read from the ResourceMetadata table.
- *Apply for all resources in the dataset checkbox* if a parent dataset is found, then the user can set the authority rule for all resources in the dataset by checking this checkbox. Instead of setting each resource one by one.
- *Priority text input*: is a value between 0 and 1.
- *Note text input* is an optional text note.
- *Recompute cache/NSM button* starts the process of computing cache or the nested set model for fast access. In the first proof-of-concept where data is not large, this is not required.
- *Add Rule button* starts the process of checking the consistency of the rule, then adds it or shows the error message if the rule causes inconsistency.
- *Cancel button* closes the console.
- *Related Authority Rules list* is a table that shows a list of authority rules in the databases which are related to the new authority. These are the authority rules on the same resource or on the same scope. They are updated while the user is filling the fields, or may be a button called Show Related Rules can refresh them. They help the

user while writing the new rule to understand the effect of the rule such as what overrides it and what does it override. The user can delete rules also from the list. The delete button starts the process of checking the consistency before deleting the rule.

The second console that we show here is the simplified end-used console (see Figure 5.11). The user can search for authority rules by giving the scope and the resource. The scope and resource are defined in the expert console. The search button retrieves the list of authority rules which are defined on the given scope and/or are defined for the given resource. The results are shown in the results table.

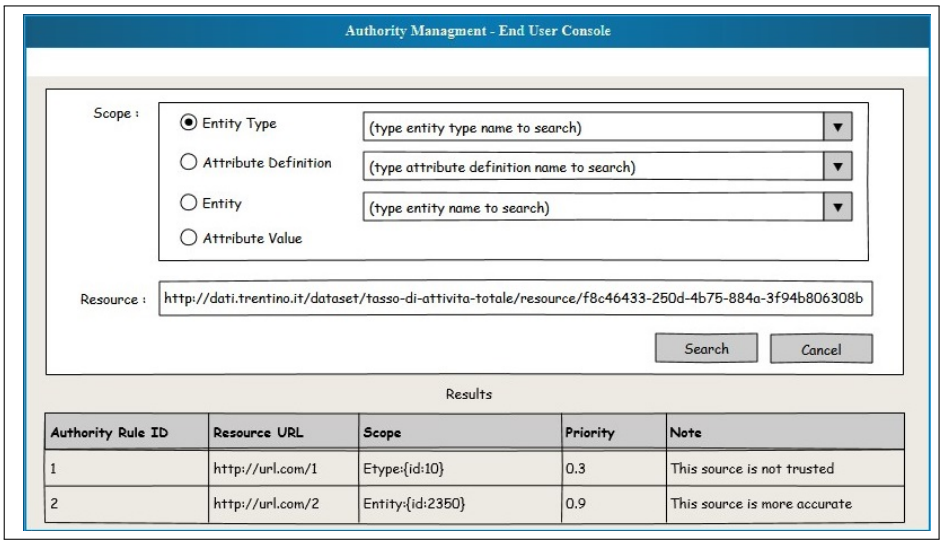


Figure 5.11: Authority Management End User Console UI

5.5.2 Provenance and Evidence visualization

Provenance and evidence visualization is the display of the provenance and evidence elements to the user. Provenance is associated with an attribute value and evidence(s) are associated with an entity. Therefore, provenance is shown with the attribute value as a meta-attribute; and evidences are shown with the entities as attributes. I.e. there is no specific UI for

showing provenance and evidence. They should appear with the entities and attribute values in their interfaces.

To make provenance and evidence appear in the UI, it must be extended to call the read operations of provenance and evidence. Evidence is shown as a normal attribute of the entity (may be with some sign that differentiate it with other attributes from the entity base). Provenance is shown as a meta attribute of an attribute value.

5.6 Use case

To validate our approach of tracing sources, this section shows a practical use case for tracing sources taken from Open Data Rise project.

This section presents the scenarios that use provenance in Open Data Rise (ODR) project. The scenarios are divided into three phases: scenarios for authority rules specifications, scenarios during the six import steps and query scenarios. An overview of provenance usage scenarios in ODR is given in Figure 5.12. There are two fail scenarios shown in red. The import process, the list of properties with their mapping, the provenance and evidence are all created in the memory of ODR. They are created in the provenance database at step 6. This allows the operations to be reversible in ODR. The permanent storage in provenance database will be only for the final elements. ODR may store the original resource files permanently or they can be stored in an external storage.

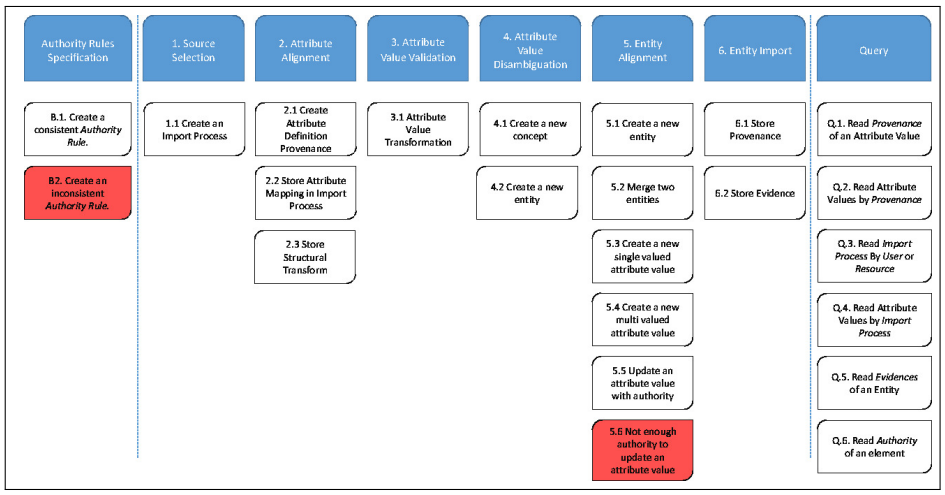


Figure 5.12: Overview of provenance usage scenarios in ODR

5.6.1 Authority Rules Specification Scenarios

(B.1) Create a consistent Authority rule: The pipeline bootstrapper wants to specify the authority of a resource. He uses the authority management console user interface (to be developed) or the HTTP API of entity base provenance to create the rule. If the created rule is consistent with other rules which are defined before, it is stored successfully in the authority rules database.

(B.2) Create an inconsistent Authority Rule: This fail scenario happens when the pipeline bootstrapper uses the authority management console user interface or the HTTP API to create an inconsistent authority rule. The back-end will discover that this rule is not consistent and will not create it. An error message will be shown to the user.

5.6.2 Import Scenarios

(1.1) Create an Import Process: Every time a user (pipeline bootstrapper or data publisher) starts the pipeline by performing the source selection step, a new import process is created in the memory of ODR (and later

stored in provenance database) automatically by the pipeline. This import process stores a reference to the selected resource, the import date and the responsible user. The resource reference will be an entity of etype distribution that has attribute definitions given in a separate report. The source dataset provenance (i.e. the first level of provenance also known as resource metadata) is stored in the resource entity.

If the distribution belongs to a new dataset, then the dataset entity is created. And if the dataset belongs to a new catalog, then the catalog entity is created. A distribution resource can be imported multiple times once, possibly each time with different columns and rows. The attribute values of the Resource entity are updated if the external meta-data are changed.

(2.1) Create Attribute Definition Provenance : During the attribute alignment step, the pipeline bootstrapper may create a new attribute definition. This attribute definition is added locally to the etype as a free attribute definition. After successfully creating the attribute definition, the user interface should seamlessly create also a knowledge base provenance for this attribute definition. The source field of the provenance will be the resource which has the new attribute definition. The validator1 field of the provenance is the user who created the attribute definition in the PKC since he validates the attribute definition before inserting it from the resource in the PKC. In this scenario, the pipeline bootstrapper (role in ODR) acts as a PKC_VALIDATOR (role in provenance). Validator2 field will be empty since this is only a local free attribute definition and it is not going to be promoted to UKC.

(2.2) Store Attribute Mapping in Import Process: During the attribute alignment step, the pipeline bootstrapper uses the ODR UI to create a mapping between the original columns in the resource and attribute definitions from an entity type. This mapping is stored in the Import Process

element in the entity base provenance module as a list of SourceProperty. If the column name is not clear, the user can add a user friendly name to the PropertyName field. If the import is partial, the columns which are not imported in this import process will have their isImported field set to false. If the user changed the mapping later, then ODR deletes the old list of SourceProperty and creates a new list with the updated mapping.

(2.3) Store Structural Transform: During the attribute alignment step, the pipeline bootstrapper may split or join columns. This structural transform is stored with the attribute value transform (see scenario 3.1) in the transform field.

(3.1) Attribute Value Transform: During the attribute value validation step, the pipeline bootstrapper or the data publisher may perform a transform on the value. This transform can be anything from a format transform to replacing the value with another new value. The transform field in the EB provenance of the attribute value is filled by ODR with an automatically generated string based on the transform. (Todo: list all the possible transforms.)

The transform field is only a string field. If multiple transforms occur, then they are combined in one string with a separator.

(4.1) Create a new concept: During the attribute value disambiguation step, the pipeline bootstrapper or the data publisher may find new concepts in the natural language strings of some attribute values. These concepts are found by running an NLP pipeline on the string values. The user may add the concepts to the local peer knowledge base. After successfully creating each concept, the user interface should seamlessly create also a knowledge base provenance for this concept. The user may also add the word and the synset for the concept. The provenance will follow the same rules of the new attribute definition provenance in scenario 2.1.

(4.2) Create a new entity: In the attribute value disambiguation step,

the pipeline bootstrapper or the data publisher may find new entities while performing vertical entity disambiguation (at column level). Most likely the value of this column will be the entity name. If this entity does not exist in the entity base, the user may create it. If the entity is created only from its name, it will have one attribute value (name) which needs provenance. The entity itself needs an evidence. The user interface should create a provenance for the attribute value as in scenario 5.5 (because the name is multi valued attribute) and an evidence for the entity as in scenario 5.1 (i.e. we ignore the column and use the whole row as evidence). An evidence uses a reference to ResourcePart which is generic and can encode a row, a value or even a part of the value such as the location of a sub string. By default, only the row is stored. If needed, we can allow the pipeline bootstrapper to configure the level of details that must be stored with the evidence.

(5.1) Create a new entity: During the entity alignment step, the pipeline bootstrapper or the data publisher may find new entities while performing horizontal entity disambiguation (at row level). An evidence is added for this new entity in ODR only (see 6.2). For creating the attribute values of this entity, see scenarios 5.3 and 5.4.

(5.2) Merge two entities: During the entity alignment step, if the attribute values from the source row are going to be merged with an existing entity then an evidence should be added to the existing entity in ODR only (see 6.2). The evidence will point to the resource part (e.g. the row) in the external source which represents the same entity.

(5.3) Create a new single valued attribute value: When creating a new attribute value for an attribute which is single valued, a provenance is created for this attribute value in ODR only (see 6.2). The provenance contains the resource part and an optional note. The authority checking is not needed because the attribute value is new.

(5.4) Create a new multi valued attribute value: When creating a new attribute value for an attribute which is mutli valued, the authority is not checked. This is because there is no overriding for an existing attribute value. The new value is added to the list of attribute values for this attribute and a provenance is created in the same way like in scenario 5.3.

(5.5) Update an attribute value with authority: When updating an existing attribute value for a single valued attribute, the authority must be checked first. If the resource which contains the new value has enough authority to override the existing attribute value, then the old value and its provenance is replaced with the new value and its provenance. At Entity Import step of the pipeline, the old value and its provenance are deleted forever from the entity and they cannot be restored. The new provenance is created in the same way like in scenario 5.3.

(5.6) Not enough authority to update an attribute value: This fail scenario happens when an attribute value of an existing entity should be updated with a new value from an external resource, but the resource which contains the new value does not have enough authority to override the existing attribute value. The existing attribute value and its provenance are not changed. The new value is not stored in the entity base and it is removed from ODR memory. This failed update should be reported to the user. The user can fix this failure, by changing the authority rules.

(6.1) Store Provenance: During the entity import step, the attribute values of the entities are created in the entity base. With each attribute value created or updated, the provenance of the attribute value is also created in the provenance module of the entity base. In scenarios 5.3, 5.4 and 5.5 the provenance is stored only in the memory of ODR and in this scenario it is permanently created in the provenance module. The pipeline user may add a note on each provenance before storing it.

(6.2) Store Evidence: During the entity import step, entities can be

created or merged with existing entities in the entity base. For each entity created or merged in the entity base, an evidence is created in the provenance module and is associated with the entity. In scenarios 5.1 and 5.2 the evidence is stored only in the memory of ODR and in this scenario it is permanently created in the provenance module.

5.6.3 Query Scenarios

(Q.1) Read Provenance of an Attribute Value: The entitypedia user wants to know the provenance of an attribute value. In the entitypedia user interface, the user can click on an attribute value and see its provenance. From the provenance, the user can see a localized, human readable description (generated by ODR) of the transform which was applied on this attribute value and any note from the importer that informs the user on how the value is imported, computed or inserted by a user. User notes will be in the language of the user who created them. Multi-lingual notes may be supported later. Also from the provenance, the user can navigate to the Import Process of this provenance and see more details such as the import date, the original resource, metadata of the original resource, original column names and column mappings. The import process is stored in the provenance database.

(Q.2) Read Attribute Values by Provenance: The entitypedia user wants to find all the attribute values that were created with a given provenance. Since the provenance refers to a single resource part, most likely there will be only one attribute value for each provenance.

(Q.3) Read Import Process By User or Resource: The entitypedia user wants to find all the import processes that were created with a given user or that was done on a specific resource. The resource can be a catalog, a dataset or a distribution. The entitypedia user can choose a user or a resource, possibly during scenario Q.1 or through a dedicated search for

users or resources, then he can list all the import processes for this user or resource.

(Q.4) Read Attribute Values by Import Process: The entitypedia user wants to find all the attribute values that were created with a given import process. The user can choose an import process, shown by scenarios Q.1 or Q.3, then he can list all the attribute values of this import process.

(Q.4) Read Evidences of an Entity: The entitypedia user wants to find the evidences of an entity. The evidences are external entities which represent the same real-world entity. In the entitypedia user interface, the user can click on an entity and see the list of evidences associated with it. The user interface may show the original resource with the evidence part highlighted if possible.

(Q.5) Read Authority of an Element: The pipeline bootstrapper or the data publisher wants to know the resources that have authority rules defined on a given attribute value, entity, entity type or attribute definition. In the entitypedia user interface, the user can click on element and choose to open the authority management console for this element.

5.7 Summary

Tracing sources is a required feature while aggregating data from different data providers. It becomes an essential requirement in the case of open data because it helps in quality assurance and in the reply of the import process. We proposed a source tracing module that extends any import process with three tools: authority, provenance and evidence. We showed how to modify an existing import process to use these tools with a use case from Open Data Rise project.

Some parts of this chapter have been published in [70].

Chapter 6

Conclusion

6.1 Thesis Summary

An entity-centric model divides the data into the entities and the language that describes them. We call the database which stores the first part an *entity base* and the database which stores the second part a *knowledge base*. In this thesis we presented an approach to build the required quality for both parts. In the web services scenarios which require combining data from different data providers, the quality certification problem is critical to the success of any aggregated entity base. This quality problem becomes more important in the domain of open data where everyone can produce or consume data freely.

The quality certification requirements in the entities base is different from the quality certification requirements in the knowledge base. The knowledge base elements are imported from external sources which can be users or resources. The elements are created either by a language translation or domain development. Two expert validators check the elements before they are finally accepted. We developed a model for tracing the knowledge base provenance of this importing and validation process.

For the entity base, we proposed a source tracing module that extends any import process with three tools: authority, provenance and evidence.

We showed how to modify an existing import process to use these tools with a use case from Open Data Rise project.

Before inserting a dataset in the entity base, entities must be imported from this dataset using a semantifying pipeline. The details of a pipeline steps are introduced with examples for each step. This pipeline can be used in many different scenarios from government, business or personal data. Any dataset can be semantified by this pipeline and the provenance will be stored automatically if the source tracing module is used with this import process.

6.2 Future Work

This thesis could be a starting point for several future directions.

- The entity import pipeline has open research problems in almost all steps. For instance, the attribute alignment step is based on schema matching which has open research issues when it comes to the special case of aligning attributes with a predefined entity types. The visualization step requires Human Computer Interaction (HCI) experts to research for innovative user interface for the imported entities.
- The knowledge base provenance model and services can be used for language development and domain development. Currently it is only validated with language development use cases. A domain development use case is needed in future to complete this part.
- Extending the PROV standards (such as PROV-DM and PROV-O) for knowledge base and entity base provenance to allow the comparison between the KB/EB provenance and the standard PROV which is required for evaluation; and also to allow interoperability with external provenance stores and visualization tools.

- The entity base provenance module is missing the *Destination*. It is source tracing tool that is the reverse of provenance. It is used to tell where an element is used after it was exported from the entity base. Pingback[1] technique can be compared with the Destination in EB provenance and an extended pingback/destination model can be proposed.

- Finding the complete list of sources that contributed to an entity (e.g. a scientific paper) by collecting the provenance of each item in it (e.g. the source of the graphs in the paper, the writer of some parts in the paper). This feature seems to be required by the community but not completely met. The entity base provenance can solve this problem using provenance of attribute values. The value of a relational attribute value can be another entity with attribute values that have provenance, hence the provenance can be queried recursively for the complete provenance list.

- From architecture point of view, we may study the possibility of collecting provenance seamlessly using aspect-oriented programming. Using aspect-oriented programming is a promising implementation strategy. Since collecting provenance seamlessly is an open engineering problem in the provenance implementation. Considering provenance as an aspect that is separated from the application logic increases modularity and simplifies the development of provenance-aware applications.

- Studying provenance in the case of crowdsourcing with entity bases. The papers [75] and [76] may be useful as examples to online adaptation of the crowdsourcing behavior. E.g. assign or not assign a specific kind of task to a user based on provenance data.

[1] http://www.w3.org/TR/prov-aq/#provenance-pingback

Bibliography

[1] Bonita Business Process Manager. http://www.bonitasoft.com/products/bonita-open-solution-open-source-bpm.

[2] CloverETL. http://www.cloveretl.com.

[3] CSV2RDF4LOD. http://logd.tw.rpi.edu/technology/csv2rdf4lod.

[4] D2R Server. http://d2rq.org/d2r-server.

[5] Data Master. http://protegewiki.stanford.edu/wiki/DataMaster.

[6] Elixir Repertoire. http://www.elixirtech.com/products/DataETL.html.

[7] Informatica Powercenter. http://www.informatica.com/us/products/enterprise-data-integration/powercenter.

[8] KNIME. http://www.knime.org/.

[9] Krextor. http://kwarc.info/projects/krextor/.

[10] LODRefine. http://github.com/sparkica/LODRefine.

[11] MapOnto. http://www.cs.toronto.edu/semanticweb/maponto/.

[12] Mapping Master. http://protege.cim3.net/cgi-bin/wiki.pl?MappingMaster.

[13] Microsoft PowerPivot, howpublished = "http://www.microsoft.com/en-us/bi/powerpivot.aspx".

[14] ODEMapster. http://neon-toolkit.org/wiki/ODEMapster.

[15] OKKAM Refine extension. http://github.com/seyyaw/okkam-refine-extension.

[16] Open Refine. http://openrefine.org.

[17] Orange. http://orange.biolab.si.

[18] Palantir. http://wwws.palantir.com.

[19] Palo. http://www.palo.net/index.php?id=6.

[20] Pentaho. http://www.pentaho.com.

[21] Pervasive Datarush. http://bigdata.pervasive.com.

[22] Poolparty Extractor. http://www.semantic-web.at/poolparty-extractor.

[23] R2R. http://wifo5-03.informatik.uni-mannheim.de/bizer/r2r/.

[24] Rapid Miner. http://rapid-i.com/content/view/181/190/lang,en/.

[25] Rattle. http://rattle.togaware.com.

[26] RDF 123. http://ebiquity.umbc.edu/project/html/id/82/RDF123.

[27] RDF Refine. http://refine.deri.ie/.

[28] RDOTE. http://rdote.sourceforge.net/.

[29] Relational.OWL. http://www.dbs.cs.uni-duesseldorf.de/RDF/.

[30] SpagoBI. http://www.spagoworld.org/xwiki/bin/view/SpagoBI/.

[31] Spotfire. http://spotfire.tibco.com.

[32] T2LD. http://ebiquity.umbc.edu/paper/html/id/480/T2LD-An-automatic-framework-for-extracting-interpreting-

[33] Tableau. http://www.tableausoftware.com.

[34] Talend. http://www.talend.com.

[35] TextPipe. http://www.datamystic.com/textpipe.html.

[36] The RDF Data Cube Vocabulary. http://www.w3.org/TR/vocab-data-cube/.

[37] TopBraid Composer. http://www.topquadrant.com/products/TB_Composer.html.

[38] Triplify. http://triplify.org/.

[39] Virtuoso Sponger. http://virtuoso.openlinksw.com/dataspace/doc/dav/wiki/Main/VirtSponger.

[40] Virtuoso's RDF Views. http://virtuoso.openlinksw.com/whitepapers/relational%20rdf%20views%20mapping.html.

[41] VisAVis. http://www.cn.ntua.gr/~nkons/.

[42] XLWrap. `http://xlwrap.sourceforge.net/`.

[43] Fausto Giunchiglia Biswanath Dutta Abed Alhakim Freihat. Approaching regular polysemy in wordnet, 2013.

[44] Abdulaziz Albatli, Lydia Lau, and Jie Xu. Application of prov model for modeling a vm overload mitigating strategy: Task eviction, 2014.

[45] Mufajjul Ali and Luc Moreau. A provenance-aware policy control framework for creating provenance-aware services, 2014.

[46] Pinar Alper, Khalid Belhajjame, Carole A. Goble, , and Pinar Karagoz. Labelflow: Exploiting workflow provenance to surface scientific data provenance., 2014.

[47] Ganbold Amarsanaa, Farazi Feroz, Reyad Moaz, Giunchiglia Fausto, and Nyamdavaa Oyundari. An experiment in managing language diversity across cultures. 2014.

[48] Z. Bao, B. Kimelfeld, Y. Li, and H. Yang. Search quality via query provenance visualization, November 18 2014. US Patent 8,892,546.

[49] Ivan Bedini, F. Farazi, David Leoni, Juan Pane, Ivan Tankoyeu, and Stefano Leucci. Open government data : Fostering innovation. Journal of eDemocracy and Open Government, 6(1):69–79, 2014.

[50] Stanislav Beran, Edoardo Pignotti, , and Peter Edwards. Interrogating capabilities of iot devices, 2014.

[51] Christian Bizer, Tom Heath, and Tim Berners-Lee. Linked Data - The Story So Far. International Journal on Semantic Web and Information Systems (IJSWIS), 5(3):1–22, MarMar 2009.

[52] JENS BLEIHOLDER and FELIX NAUMANN. Data fusion. 2008.

[53] Jens Bleiholder and Felix Naumann. Data fusion. ACM Comput. Surv., 41(1):1:1–1:41, January 2009.

[54] Katrin Braunschweig, Julian Eberius, Maik Thiele, and Wolfgang Lehner. The state of open data limits of current open data platforms, 2012.

[55] Oleksiy Chayka. Three case studies for understanding, measuring and using a compound notion of data quality with emphasis on the data staleness dimension. doctoral thesis., 2012.

[56] James Cheney and Roly Perera. An analytical survey of provenance sanitization, 2014.

[57] Victor Cuevas-Vicenttin, Bertram Ludascher, and Paolo Missier. Rinke hoekstra and paul groth., 2014.

[58] Saumen Dey, Sven Koehler, Shawn Bowers, and Bertram Ludaescher. Optimizing data lineage queries using static workflow analysis, 2014.

[59] Laura Dragan, Markus Luczak-Roesch, Elena Simperl, Bettina Berendt, and Luc Moreau. Crowdsourcing data citation graphs using provenance, 2014.

[60] Mikalai Yatskevich Fausto Giunchiglia and Pavel Shvaiko. Semantic matching: Algorithms and implementation, 2007.

[61] Hugo Firth and Paolo Missier. Provgen: generating synthetic prov graphs with predictable structure., 2014.

[62] J. Freire, C.T. Silva, S.P. Callahan, C.E. Scheidegger, and H.T. Vo. Enabling provenance management for pre-existing applications, May 29 2012. US Patent 8,190,633.

[63] Luiz Gadelha and Marta Mattoso. Applying provenance to protect attribution in distributed computational scientific experiments, 2014.

[64] A. Ganbold, F. Farazi, M. Reyad, O. Nyamdavaa, and F. Giunchiglia. Managing language diversity across cultures: the english-mongolian case study. International Journal on Advances in Life Sciences, 6(3&4):167–176, 2014.

[65] Daniel Garijo, Yolanda Gil, and Andreas Harth. User requirements for geospatial provenance, 2014.

[66] David George. Understanding structural and semantic heterogeneity in the context of database schema integration.

[67] Devarshi Ghoshal and Arun Chauhan andand Beth Plale. Regenerating and quantifying quality of benchmarking data using static and dynamic provenance, 2014.

[68] Fausto Giunchiglia and Biswanath Dutta. Dera: A faceted knowledge organization framework, 2011.

[69] Fausto Giunchiglia, Vincenzo Maltese, Feroz Farazi, and Biswanath Dutta. Geowordnet: A resource for geo-spatial applications. In Proceedings of the 7th International Conference on The Semantic Web: Research and Applications - Volume Part I, ESWC'10, pages 121–136, Berlin, Heidelberg, 2010. Springer-Verlag.

[70] Fausto Giunghiglia and Moaz Reyad. Provenance in open data entity-centric aggregation. In Bertram Ludscher and Beth Plale, editors, Provenance and Annotation of Data and Processes, volume 8628 of Lecture Notes in Computer Science, pages 232–234. Springer International Publishing, 2015.

[71] Alasdair JG Gray. Dataset descriptions for linked data systems. Internet Computing, IEEE, 18(4):66–69, 2014.

[72] Farshad Hakimpour and Andreas Geppert. Resolving semantic heterogeneity in schema integration: an ontology based approach. In Proc. of the Intl. Conf. On Formal Ontologies in Information Systems (FOIS-2001), ACM, pages 297–308. ACM Press, 2001.

[73] Rinke Hoekstra and Paul Groth. Prov-o-viz - understanding the role of activities in provenance., 2014.

[74] Trung Dong Huynh, Mark Ebden, Sarvapali Ramchurn, Stephen Roberts, and Luc Moreau. Data quality assessment from provenance graphs, 2014.

[75] Amir Sezavar Keshavarz, Trung Dong Huynh, , and Luc Moreau. Provenance for online decision making, 2014.

[76] Amir Sezavar Keshavarz, Trung Dong Huynh, and Luc Moreau. Provenance for online decision making, 2014.

[77] Timothy Lebo, Patrick West, , and Deborah L. McGuinness. Walking into the future with prov pingback: An application to opendap using prizms, 2014.

[78] Marta Mattoso, Jonas Dias, Flavio Costa, Daniel Oliveira, and Eduardo Ogasawara. Experiences in using provenance to optimize the parallel execution of scientific workflows steered by users, 2014.

[79] Paolo Missier, Jeremy Bryans, Carl Gamble, Vasa Curcin, , and Roxana Danger. Provabs: model, policy, and tooling for abstracting prov graphs, 2014.

[80] Leonardo Murta, Vanessa Braganholo, Fernando Chirigati, David Koop, , and Juliana Freire. noworkflow: Capturing and analyzing provenance of scripts, 2014.

[81] Tom De Nies, Robert Meusel, Dominique Ritze, Kai Eckert, Anastasia Dimou, Laurens De Vocht, Ruben Verborgh, Erik Mannens, and Rik Van de Walle. A lightweight provenance pingback and query service for web publications., 2014.

[82] Juan Pane. Distributed identity management. doctoral thesis., 2012.

[83] Massimo Paolucci and Katia Sycara. Autonomous semantic web services, 2003.

[84] Quan Pham, Tanu Malik, and Ian Foster. Auditing and maintaining provenance in software packages, 2014.

[85] Manolis Stamatogiannakis, Paul Groth, and Herbert Bos. Looking inside the black-box: Capturing data provenance using dynamic instrumentation, 2014.

[86] Curt Tilmes. Formal provenance representation of the data and information supporting the national climate assessment, 2014.

[87] M.B. VELASCO. Use and enforcement of provenance and lineage constraints, January 17 2013. US Patent App. 13/183,850.

[88] C.C. White, M.W. Thomas, and A.J. O'Leary. Managing provenance of digitally signed data in user editable records, December 10 2009. US Patent App. 12/136,040.

[89] Adianto Wibisono, Peter Bloem, Gerben De Vries, Paul Groth, Adam Belloum, and Marian Bubak. Generating scientific documentation for computational experiments using provenance, 2014.

[90] J. Zhao, F. Sun, C. Torniai, A.B. Bakshi, and V.K. Prasanna. System and method for data provenance management, September 10 2013. US Patent 8,533,152.

[91] Jun Zhao, Honghan Wu, and Jeff Z. Pan. Towards query generation for prov-o data., 2014.