ICT Seventh Framework Programme (ICT FP7)

Grant Agreement No: 318497
Data Intensive Techniques to Boost the Real – Time Performance of Global Agricultural Data Infrastructures

SemaGrow

D3.3.2: Techniques for Content Classification & Ontology Evolution

<table>
<thead>
<tr>
<th>Deliverable Form</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Project Reference No.</strong></td>
</tr>
<tr>
<td><strong>Deliverable No.</strong></td>
</tr>
<tr>
<td><strong>Relevant Work Package:</strong></td>
</tr>
<tr>
<td><strong>Nature:</strong></td>
</tr>
<tr>
<td><strong>Dissemination Level:</strong></td>
</tr>
<tr>
<td><strong>Document version:</strong></td>
</tr>
<tr>
<td><strong>Date:</strong></td>
</tr>
<tr>
<td><strong>Authors:</strong></td>
</tr>
<tr>
<td><strong>Document description:</strong></td>
</tr>
</tbody>
</table>
# Document History

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Author (Partner)</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Draft v0.1</td>
<td>07/07/2015</td>
<td>UNITOV</td>
<td>Initial draft</td>
</tr>
<tr>
<td>Draft v0.4</td>
<td>10/07/2015</td>
<td>UNITOV</td>
<td>Added experimentation with FAO on AgroIE and custom ranges</td>
</tr>
<tr>
<td>Draft v0.7</td>
<td>13/07/2015</td>
<td>UNITOV</td>
<td>More details on the custom ranges</td>
</tr>
<tr>
<td>Draft v0.8</td>
<td>14/07/2015</td>
<td>UNITOV</td>
<td>Minor revisions on the document</td>
</tr>
<tr>
<td>Final v1.0</td>
<td>31/07/2015</td>
<td>UNITOV, NCSR-D</td>
<td>Internal Review / Document Finalization</td>
</tr>
</tbody>
</table>
EXECUTIVE SUMMARY

This deliverable depicts possible evolution scenarios for the SemaGrow federated datasets and provides methodological and technological solutions for supporting them. The evolution here is expressed in terms of the acquisition of new content (expressed through different media, but generally addressable as unstructured content) to enrich with new information the existing datasets. The deliverable reports “classification” but is actually meant to cover in a broader sense the triplification of unstructured content. Evolution is also addressed in terms of changes at the level of vocabulary, and is thus convoluted into a manageable workflow providing a neat separation between the processes of knowledge acquisition, and their possible and efficient application to different ontology vocabularies, or to evolutions of the same ones.

As for the alignment techniques proposed in T3.2, the results of this research and of its applied development find an immediate application in the context of the federated scenarios as depicted in SemaGrow, but represent at the same time a viable and widely acceptable solution for all cases related to acquisition of RDF data from unstructured content.

In the first deliverable D3.3.1, we provided the details about the framework proposed by UNITOV, and described how this had been refined and improved to capture the large extent of automatism on a distributed scale, as demanded by SemaGrow.

In this second edition of the deliverable, we detail the improvements to past work and the concluded efforts that led to interesting results for the SemaGrow scenario, as well as productive interactions with other connected projects such as AgInfra.
# TABLE OF CONTENTS

LIST OF FIGURES .................................................................................................................. 5

LIST OF TABLES ...................................................................................................................... 6

LIST OF TERMS AND ABBREVIATIONS ............................................................................. 6

1. INTRODUCTION .................................................................................................................. 8
   1.1 Purpose and Scope .......................................................................................................... 8
   1.2 Approach for Work Package and Relation to other Work Packages and Deliverables .......... 8
   1.3 Methodology and Structure of the Deliverable ............................................................... 8

2. CONTENT ACQUISITION/CLASSIFICATION AND ONTOLOGY EVOLUTION: STATE OF THE ART ................. 10

3. CODA: COMPUTER-AIDED ONTOLOGY DEVELOPMENT ARCHITECTURE ................................................... 11
   3.1 UIMA: Unstructured Information Management Architecture .............................................. 11
   3.2 CODA Analysis Engine .................................................................................................. 11
   3.3 PEARL: The Projection Language .................................................................................... 14
      3.3.1 Language Overview ..................................................................................................... 14
      3.3.2 Operational Semantics ............................................................................................... 15
      3.3.3 A Comparison of PEARL with Other Existing Approaches ........................................ 17

4. USE CASES AND WORK CONDUCTED IN THE LAST 15 MONTHS ..................................................... 19
   4.1 Improvements to the main framework ............................................................................... 19
      4.1.1 General framework improvements .............................................................................. 19
      4.1.2 PEARL Improvements ............................................................................................... 20
      4.1.3 Converters .................................................................................................................. 20
      4.1.4 CODA Integration with other systems ......................................................................... 20
   4.2 Acquisition from Raw Natural Language Content. Relationship-Extraction ............................ 21
   4.3 Acquisition from Semi-structured content. Spreadsheet processing .................................. 23
      4.3.1 Improvements ............................................................................................................ 24
      4.3.2 Adoption of the system ............................................................................................... 25
   4.4 Integration of CODA inside Semantic Turkey: Custom Ranges ............................................ 29

5. REFERENCES ....................................................................................................................... 35
LIST OF FIGURES


Figure 2. A PEARL rule example.

Figure 3. PEARL execution flow.

Figure 4. Comparison of integration approaches.

Figure 5. AgroIE: content acquisition inside a CODA-powered Semantic-Turkey

Figure 6. Spreadsheet-import and PEARL generation inside Semantic-Turkey with Sheet2RDF

Figure 7. Spreadsheet-import and PEARL generation inside Semantic-Turkey with Sheet2RDF

Figure 8. DB export of SoilDB into the Excel file which has been fed to Sheet2RDF for triplification

Figure 9. Visual representation of an extract of the CRA Soil Terms vocabulary in the Vocbench platform: sub-tree under the concept “Soil Body”

Figure 10. Visual representation of the agINFRA Resource Types KOS in the Vocbench

Figure 11. selection of custom ranges and association to ontology properties

Figure 12. PEARL rule used in ST for driving the form generation for the acquisition of a reified skos:definition

Figure 13. editing of a Custom Range in Semantic Turkey

Figure 14. the generated form for entering values for a reified skos definition

Figure 15. Visualization of the reified SKOS definition in the editor panel of ST
## LIST OF TERMS AND ABBREVIATIONS

<table>
<thead>
<tr>
<th>Term/Abbreviation</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Content Analytics</td>
<td>The processing of heterogeneous information sources (traditionally intended as, but not necessarily limited to, text content) to elicit structured information to be stored according to a given data scheme (in our specific scenario, an ontology vocabulary).</td>
</tr>
<tr>
<td>Graph Store</td>
<td>A mutable repository of RDF graphs managed by one or more services</td>
</tr>
</tbody>
</table>
| LOD | Linked Open Data  
Also often, found as a shortened version of “LOD Cloud”. |
| LOD Cloud | All the data sources of interconnected linked open data in the web. Currently, there exists a certain amount of connectors, such as datahub\(^1\), where LOD publishers can register their data for maximizing visibility of their content. The presence of these connectors has been debated, as they seem to contradict the distributed and decentralized nature of the (Semantic) Web. Total conformance to LOD standards and best practices should guarantee the possibility to perform a “follow-you-nose” navigation of data on the Web under the same preconditions of traditional web content. Nonetheless, these hubs and connectors actually support the diffusion and visibility of data repository in this transient period, where best practices for content publishing and access have not always been followed by publishers, nor have they been definitely assessed – in all their aspects – as standards. |
| Named Graph | Named graphs are a key concept of Semantic Web architecture in which a set of Resource Description Framework statements (a graph) are identified using a URI, allowing descriptions to be made about that set of statements such as context, provenance information or other such metadata. |
| Ontology | An “explicit specification of conceptualization” [1]. Informally, a working model of entities and interactions in some particular domain of knowledge or practices. Ambiguity may arise, especially in the context of Ontology Alignment, as the word Ontology is used to generally refer to any domain description modelled after any of the knowledge representation models that fall in the category of ontology modelling languages. However, following standardization by W3C, “ontologies” need to be distinguished from “thesauri” and “concept schemes”, which are shallower and less formal descriptions of domains usually adopted for text categorization and indexing tasks. The distinction has got sharper when W3C recommended a dedicated language (SKOS) for modelling thesauri, concept schemes and terminologies, to be adopted in place of OWL, which is now exclusively dedicated to more formal description of domains with a strong data oriented scope. However, many compound terms, such as “Ontology Alignment”, which are older than SKOS, are still used, for historical continuity, to address the problem of model and data integration across any of these languages. |
| OWL | The OWL Web Ontology Language, informally OWL, is an ontology language for the Semantic Web with formally defined semantics. OWL ontologies provide |

---

\(^1\) [http://datahub.io/group/lodcloud](http://datahub.io/group/lodcloud)
<table>
<thead>
<tr>
<th>Term/Abbreviation</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>RDF</td>
<td>The Resource Description Framework (RDF) is a W3C Recommendation [2] for a language for representing information about resources in the World Wide Web. RDF statements are triples; data entities composed of subject, predicate, and object.</td>
</tr>
<tr>
<td>SKOS</td>
<td>SKOS is a specification supporting the use of knowledge organization systems (KOS) such as thesauri, classification schemes, subject heading lists and taxonomies within the framework of the Semantic Web</td>
</tr>
<tr>
<td>SKOS-XL</td>
<td>SKOS-XL defines an extension of SKOS, providing additional support for describing and linking lexical entities</td>
</tr>
<tr>
<td>SPARQL</td>
<td>SPARQL is a W3C Recommendation for expressing queries across diverse data sources, whether the data is stored natively as RDF or viewed as RDF via middleware. The results of SPARQL queries can be result sets or RDF graphs.</td>
</tr>
<tr>
<td>SPARQL endpoint</td>
<td>A SPARQL endpoint is a conformant SPARQL protocol service. A SPARQL endpoint enables users (human or other) to query a knowledge base via the SPARQL language. Results are typically returned in one or more machine-processable formats.</td>
</tr>
<tr>
<td>Triple Store</td>
<td>A triplestore is a purpose-built database for the storage and retrieval of (RDF) triples. Following the addition of named graphs to the RDF standard, a new name: “Graph Store” has been coined, and a protocol has been established for these kind of data repositories.</td>
</tr>
<tr>
<td>UIMA</td>
<td>UIMA (Unstructured Information Management Architecture) is both an architecture and an associated framework supporting development and orchestration of analysis components for the extraction of information from unstructured content. UIMA is not limited, though naturally bound, to text analytics, and can be potentially adopted for processing any kind of media</td>
</tr>
</tbody>
</table>
1. INTRODUCTION

1.1 Purpose and Scope

This report depicts evolution scenarios for the Semagrow federated datasets and provides an overall view and some insights on the methodological and technological solutions that the consortium has proposed for supporting them. The “ontology evolution” expression here (with “ontology” being intended in the broader meaning of dataset, and mostly SKOS thesaurus in SemaGrow specific case) is expressed in terms of the acquisition of new content (represented through different media, but generally addressable as unstructured content) to enrich with new information the existing datasets.

The other keyword in the deliverable name is “classification”, actually intended to cover in a broader sense the RDF triplification of unstructured content. This may thus include, as a specific case, classification of documents as a whole, i.e. associating documents to SKOS concepts, seen as “topics”, but involves, in the more general case, the recognition and extraction of facts from documents and media in general, in order to produce new information in the target datasets.

Ontology evolution also addresses changes at the level of vocabulary, which are then convoluted into the requirement of a manageable workflow providing a neat separation between the process of knowledge acquisition, and its possible and efficient applications to different ontology vocabularies, or to evolutions of the same ones.

Specific care has been taken to verify the applicability of the suggested approach to the specific SemaGrow federated scenario; however, attention has been paid as well to producing solutions that are of large benefit with respect to the general necessity of content acquisition for evolving ontologies.

In the first release of this deliverable (D3.3.1, M18), we provided the details about the framework proposed by UNITOV, and described how this has been refined and improved to capture the large extent of automatism on a distributed scale, as demanded by SemaGrow.

In this second and final edition of the deliverable, we described the further improvements that the system received during the last 15 months of the project, especially after gathering feedback from real use of the system, a few applications that have been built on top of it, and its application to real case scenarios provided by SemaGrow and its connected project AgInfra (http://aginfra.eu).

1.2 Approach for Work Package and Relation to other Work Packages and Deliverables

The document has been developed as a living document, in which the proposed acquisition system’s architecture and related tools have been documented as they evolved from new insights gained during the project.

The input of this work package is clearly represented by:

1) The data and knowledge models that will be made available in SemaGrow (Task 2.2: Data Requirements & Preparation).

2) Unstructured content that has been made available by the partners, either produced internally (and in need of being classified/analyzed) or collected from the Web, as considered of interest for evolving the aforementioned datasets.

3) The general SemaGrow architecture (Task 2.3: Large Scale Distributed Architecture of WP2), for positioning the range of intervention and infrastructural entry points for the technical solutions provided by this work package.

1.3 Methodology and Structure of the Deliverable

This is the last (D3.3.2) of the two issues of the deliverable related to Content Evolution and Classification. As previously explained, this new version builds on top of the first one, and completes it as a self-consistent document, providing all the details about the improvements done and new results obtained in the last 15 months after D3.3.1 was initially released.

In the remainder of this document, we first describe the typical needs and scenario for ontology/content evolution, the current state of the art and its limitations. We then introduce our proposed framework, CODA, and the improvements we have been targeting in order to meet specific SemaGrow requirements. We then provide more insights on CODA...
components, and the associated maintenance tools (Section 3). Section 4 concludes by describing specific applications based on CODA developed in the context of this project and applied to SemaGrow use cases, and the results obtained.
2. Content Acquisition/Classification and Ontology Evolution: state of the Art

Efficient Information Management and Information Gathering are becoming extremely important to derive value from the large amount of available information. While the uptake of Linked Data [3] promoted uniform standards for the publication of information as interlinked datasets, the Web still consists mainly of unstructured content. According to our internal informal audit among the various user partners of the SemaGrow consortium, their organizations have plethora of internally produced/collected content sources that are obviously relevant for their datasets. They would aim at least to link these sources to their datasets on a per-topic basis, but more precise content extraction and triplification is also much desired, though none has the knowhow to carry on it nor, in the best case, considers it as an easy task to manage. Indeed, dealing with this heterogeneous content asks for coordinated capabilities of several dedicated tools. Development of knowledge acquisition systems today largely requires non-trivial integration effort and the development of ad hoc solutions for tasks that could be better defined and channeled into an organic approach.

Platforms such as GATE [4] and UIMA [5] provide standard support for content analytics, while they completely delegate to developers tasks concerning data transformation and publication. There have been a few attempts at completing content analytics architectures with facilities for the generation of RDF data, e.g. the RDF UIMA CAS Consumer\(^2\) and Apache Stanbol\(^3\). The former is a UIMA component consuming the analysis metadata to generate RDF statements, while the latter focuses on semantic content management. These projects share the same approach, which consists in a vocabulary-agnostic serialization of metadata inferred by analytics. However, Stanbol provides a component, called Refactor Engine\(^4\), which can refactor the extracted triples, possibly to target a user-chosen vocabulary. The Refactor Engine performs RDF graphs transformations to specific target vocabularies or ontologies by executing rules expressed in a dedicated syntax, which is then grounded over SPARQL CONSTRUCT.

A completely different approach is the one followed by semantic search engines such as Eqentia [6] or Evri [7], and Information Extraction services such as OpenCalais [8] and Zemanta [9]. These services provide facilities for producing knowledge modeled according to open standards. However, users may not extend them, e.g. by providing new content extractors and annotators, and there is thus no way to address specific exigencies both in terms of the content to be analysed and of the target models to be enriched/populated.

In our opinion, the above approaches lack an overall perspective on the task, as well as a proposal of a gluing architecture supporting the triplification process as a whole.

In the context of Task 3.3, we discussed yet from the SemaGrow kick-off the benefits of adopting CODA (Computer-aided Ontology Development Architecture), an architecture based on standard technologies (such as UIMA, OSGi\(^5\), and OBR\(^6\)) and data models (e.g. RDF, OWL [10] and SKOS [11]), which defines a comprehensive process for the production of semantic data from unstructured content. This process includes content analysis, generation of RDF out of extracted information, and involvement of humans for validation and refinement of the produced triples.

The CODA architecture has been designed by UNITOV, and a specific platform is being developed around it for supporting the grounding of CODA compliant systems. The objective to be fulfilled inside SemaGrow is to refine and improve the architecture having in mind the specific scenarios of the SemaGrow content providers federation.

\(^3\) http://stanbol.apache.org/
\(^4\) https://stanbol.apache.org/docs/trunk/components/rules/refactor.html
\(^5\) http://www.osgi.org/Specifications/HomePage
\(^6\) http://felix.apache.org/site/apache-felix-osgi-bundle-repository.html
3. CODA: Computer-Aided Ontology Development Architecture

CODA extends UIMA with specific capabilities for populating RDF datasets and enriching ontologies with information mined from unstructured content. We describe here the CODA Architecture, its main features, and the improvements that we brought in the context of the SemaGrow project. We start this section by providing also a larger introduction to UIMA, later explaining its interfacing with CODA.

3.1 UIMA: Unstructured Information Management Architecture

UIMA offers an architecture (OASIS Standard in 2009) and an associated framework (released by IBM as open source in 2004, then donated in 2006 to the Apache Software Foundation [12]), supporting development and orchestration of analysis components – called Analysis Engines (AE) – for the extraction of information from unstructured content. UIMA is not limited, though naturally bound, to text analytics, and can be potentially adopted for processing any kind of media.

In UIMA, a CAS (Common Analysis Structure) [13] is the primary data structure that Analysis Engines use to represent and share analysis results. A CAS contains both the information to be analyzed (called Subject of Analysis, or Sofa) and the extracted metadata, represented as typed feature structures [14] constrained by a model called Type System (TS) in the UIMA terminology. A specific kind of metadata (annotations) contains references to the region of the Sofa from which it was extracted. A total order is defined over annotations, usually reflecting the physical position of the referenced regions within the Sofa. A metadata index is maintained inside the CAS for iterating over annotations in a specified order.

Usually, a UIMA workflow consists in a simple pipeline whose last element is an Analysis Engine that consumes the extracted metadata for some purpose (e.g. to augment a structured source of information). Being focused on a general architecture, UIMA does not provide any implementation targeting a specific source, such as relational databases or RDF knowledge bases.

In Figure 1, we depict the architecture of a typical application based on CODA. Our primary contribution with respect to the standard UIMA pipeline is the introduction of a concrete Analysis Engine (CODA Analysis Engine) targeting the evolution of RDF datasets (Target RDF Dataset in the figure) where RDF datasets consist of both knowledge bases (data) and ontologies (schemes). The Analysis Engine delegates domain specific tasks to external components (see, for instance, converters, later in sections 3.2 and 3.3). In the context of SemaGrow, we pushed forward the concept of reusable components according to the same principles of the web of data, by developing a component discovery and automatic plug-in mechanism. CODA components can thus be hosted on remote repositories, be automatically discovered on the Web and be deployed into the local repository of a CODA installation (CODA Local Repository).

Ideally, in SemaGrow, we envisage the development of a CODA repository dedicated to Agriculture-related concepts. As everything in SemaGrow, this approach is not opposed to the larger view of a heterogeneous and completely distributed web of data, but aims at the concrete development of an open scenario of federated data providers, supplying also facilities and services specifically targeted towards the agriculture domain.

3.2 CODA Analysis Engine

CODA provides dedicated components and their orchestration for projecting UIMA analysis results into the Target RDF Dataset, while exploiting UIMA for the analysis of unstructured information. The main task of CODA is to project UIMA metadata contained in a CAS into user defined RDF graph patterns based on a vocabulary of choice. In performing this task, CODA integrates the fully-fledged API (Application Programming Interface) provided by UIMA for accessing the subject of analysis with its range of specific functionalities for transforming the metadata. Actually, any UIMA pipeline can be extended with the Analysis Engine wrapping CODA, so that at the end of the processing the metadata extracted so far can be projected to RDF. Additionally, existing UIMA-powered applications may deal with CODA in a uniform manner, since deployment, management and execution of UIMA components follow known conventions.

The projection of UIMA metadata to RDF triples is a complex process including the following activities:

- Select relevant information from within the CAS;
- Turn UIMA metadata into RDF nodes;
Build an RDF graph from a user defined graph pattern and the prepared RDF nodes.

For supporting the user in the specification of this process, CODA provides a rule-based pattern matching and transformation language, called PEARL (see next section for more details about its syntax and semantics).

The PEARL Execution Engine is the component implementing the PEARL operational semantics. This engine orchestrates the triplification process, executing PEARL projection rules against each CAS produced by the UIMA workflow. These rules specify in a declarative way how metadata conforming to a given type system (TS) are related to RDF triple patterns using a given vocabulary. PEARL syntax covers UIMA metadata selection and RDF graph construction patterns. The construction of RDF nodes out of UIMA metadata is a highly varied task, possibly depending on domain and application specific
procedures for composing URIs and literals, or on external services for identity resolution (such as OKKAM [15]). In order to preserve its simplicity, the PEARL syntax provides specific extension points for delegating this task to external components, called converters.

A dedicated rule set is sufficient to handle most usage scenarios, without any coding activity in a general-purpose programming language, such as Java. Very specific tasks can be accomplished through converters implemented in Java using the API provided by CODA. Still, most of the transformation may be specified in PEARL using transformation rules that, in turn, delegate some of the work to the converters.

In the first version of CODA, a specific set of operators for lifting extracted data into RDF resources had been made available. When examining possible use cases for SemaGrow, we took into consideration the possibility of allowing the system to be extended with components developed for solving specific needs, thus a first reengineering of the platform went in that direction, and implied applying changes to the PEARL syntax, to allow for the presence of unforeseen operators, much in the spirit of SPARQL. On a second stage, we are further improving CODA with an advanced publish & discovery mechanism based on the same principles of linked open data. In the next roll out of CODA, converters will be indirectly referred by means of a URI (see section on Language Overview) which identifies the desired behavior (a contract) instead of a concrete implementation. A contract thus identifies a set of functionally equivalent converters, possibly differing for non-functional properties, such as resource consumption, performance with respect to the task, or even licensing terms. The exact behavior represented by a contract is expressed in terms of specifications in natural language (as usual in contracts for service or methods), provided through standard metadata properties (e.g. dcterms: description) and it is actually the URI of the contract which provides the sole semantic anchor. For instance, the contract <http://art.uniroma2.it/coda/contracts/default> is described as “the procedure invoked by default for transforming a UIMA value into a valid RDF term”. In fact, the input/output behavior of contracts is difficult to express in a formal language. Therefore, we are unable to exploit approaches for matching semantic web services [16], based on annotations with respect to ontologies and formal preconditions and post-conditions. However, approaches for discovering (semantic) web services through natural language descriptions might be extended to CODA contracts, as well. Influenced by the web services architecture [17], the distinction between contracts and converters helps the reuse of projection specifications, since an application can bind the contracts to the converters best fitting its own requirements.

The PEARL execution logic is kept separate from the contract resolution process as contract references are resolved into suitable converters by the Component Provider. Converters are bundled complying with the OSGi (formerly known as the Open Services Gateway initiative) specification [18] and stored in repositories organized according to the OBR (OSGi Bundle Repository) architecture [19]. OBR repositories maintain metadata about the hosted bundles including their name, version, provided capabilities and requirements. A CODA converter is advertised on OBR repositories as a service capability, which holds the contract URI, and the Java interface for interrogating the converter. OBR requirements are instead populated with the non-functional properties of the converters.

The Component Provider follows a two-step procedure for resolving contracts into suitable converters. At first it uses the OBR Client to access a known set of OBR repositories (starting from the CODA Local Repository) looking for a candidate whose metadata match the contract. If no candidate is found, the Component Provider relies on its Discoverer module to explore the Web looking for additional repositories. The Discoverer exploits the fact that PEARL specifications are self-describing [20] and, moreover, grounded in the Web of Data, since the required contracts are mentioned through dereferenceable URIs [21]. In compliance with the Linked Data principles, the Discoverer obtains through those URIs an RDF description of the contract, including a list of authoritative repositories of known implementations. This architecture enables autonomous configuration of CODA systems, disburdening the user from manual settings prior to the execution of a PEARL projection. This is especially valuable when reusing PEARL documents written by third parties, as in the open and distributed scenario described in [22].

The execution of a PEARL specification against a given CAS produces a set of triples that are eventually committed to an RDF triple store. Also, both the core PEARL syntax and the converters may refer to the underlying Target RDF Dataset, for closing a feedback loop around the Target RDF Dataset. The interaction with a triple store is mediated by the OWL ART API [23], an open RDF middleware providing implementations for different triple store technologies.
3.3 PEARL: The Projection Language

PEARL (ProjEction of Annotations Rule Language) is a pattern-matching and transformation language for the triplification of UIMA metadata. Being specifically tailored to this purpose, the language allows concise yet powerful specifications of the transformation. An initial specification of PEARL was presented in [24]. This deliverable introduces the reviewer to PEARL, and includes the latest changes brought in the context of SemaGrow, thus no preliminary reading is necessary. A detailed comparison of PEARL with alternative approaches concludes this section, in order to show the advantages of our solution.

3.3.1 Language Overview

A PEARL Document mainly consists of rules driving the UIMA Metadata Projection process. A rule is introduced by the keyword rule (optionally preceded by the keyword lazy), followed by the matched UIMA type and, optionally, by an identifier and a set of dependencies on other rules. A non-lazy rule (a rule not introduced by the keyword “lazy”) is invoked for each annotation from the index that matches the UIMA type specified in its declaration. Due to their triggering condition, such rules are also called matched rules. A lazy rule is executed only if it is invoked explicitly by a matched rule through a binding in its bindings section. Lazy rules are syntactically constrained, as they can only define a nodes section for the generation of RDF terms out of the extracted information. Therefore, matched rules may be decomposed though bindings to lazy rules, thus fostering the modularization of a PEARL rule-set.

In Figure 2, a simple PEARL rule for generating descriptions of books (based on the BIBO and FOAF vocabularies) - from information extracted by UIMA - is shown. The nodes section uses a feature path (a sequence of features inside a feature structure) to select relevant UIMA metadata from the matched annotation which will be converted to RDF nodes. These generated nodes are assigned to placeholders. For instance, in the rule in Figure 2, the UIMA feature it.uniroma2.Book:author is projected both as an RDF named resource (URI) and as a string typed literal. In the first case a URI (bookAuthor) is created after the feature content, and will be used as the resource identifying the extracted author, while in the second case a literal value (authorName) is generated to populate a property of the author (as specified in the graph section of the rule). Default conversion heuristics are applied, and are inferred on the basis of the specified node type. For instance, if the type of the target is URI, the feature value is first “cleaned”, to remove characters that are incompatible with the URI standard, and then used as a local name and concatenated to the namespace of the target ontology to create an URI. Optionally, dedicated converters (see end of the section above) may be declared to operate the transformation. In the example, a URI is generated from the extracted ISBN (International Standard Book Number), following the ISBN-A specification [25], by invoking a converter compliant to the cdbk:isbn contract, and is assigned to the bookISBN placeholder. An optional section, similar to nodes, and called bindings, allows to declare complex placeholders as objects referencing placeholders declared in lazy rules (an example on the use of bindings and lazy rules is provided later in the next section).

The graph section defines a graph pattern [26] that is used as a template for the instantiation of RDF triples. The graph pattern is dynamically instantiated with RDF nodes generated in the nodes and where sections. Inside a graph pattern, placeholders originated in the nodes section are identified through the prefixed symbol “$”, while variables grounded in the where section, are prefixed with the symbol “?”. When a placeholder and a variable have the same name, they could be related through the fallback mechanism described in section “Operation Semantics”.

Page 14 of 37
The *where* section contains a graph pattern, which is matched against pre-existing information in the Target RDF Dataset, for binding variables to matching RDF nodes by means of unification. The purpose of this section is to link newly extracted data with information that is already present in the target ontology. In the example, the graph pattern in the *where* section is composed of a single triple characterized by a variable in the subject, the property foaf:name as the predicate, and the placeholder authorName as the object. The placeholder has been initialized with a string typed literal in the nodes section, and is thus used in combination with the ground predicate to retrieve a substitution for the variable author from the Target RDF Dataset.

The key distinction between placeholders and variables lies in how they are initialized: the former with RDF nodes generated from UIMA annotations (*nodes* section), the latter with existing resources retrieved from the target semantic repository through pattern matching (*where* section).

### 3.3.2 Operational Semantics

The *Pearl Execution Engine* iterates over the UIMA annotations according to the order provided by the default annotation index. For each annotation, all matched rules that apply to the annotation type are executed to produce RDF triples. These triples will be committed after a validation phase. The execution flow of a matched rule is shown in Figure 3: each row of the table represents the assignment status of placeholders and variables at the start of each phase. The execution starts with a pre-processing routine, by parsing and replacing all prefixes in the document with their assigned namespaces. The *bindings* section, if present, contains a collection of bindings, each one instructing the engine to execute a lazy rule against the current UIMA annotation. Such a rule generates RDF nodes and assigns them to placeholders, which can be referenced within the caller rule through the binding symbolic name. Later, in section “A Use Case: Evolving AGROVOC”, we provide an example combining lazy rules and bindings.

Execution then passes to the *nodes* section by initially ensuring that all referenced *converters* are deployed in the CODA Local Repository (or, in negative case, by retrieving them from the Web, as described in section on *CODA Analysis Engine*).
After all the machinery is ready for the transformation of UIMA metadata, RDF nodes are generated and assigned to placeholders.

Ground placeholders are then forwarded to the next step that is the resolution of the where section, in which variables are assigned values through unification with the Target RDF Dataset. The application of both graph patterns from the where and graph sections may fail. In the first case, the where graph is matched against the Target RDF Dataset in much the same way of a SPARQL where condition. In PEARL, a failed match is always considered as having a result-set consisting of a single tuple with all variables left unbound, thus the rule is not considered to be failed, its execution continues, but the variables declared in the where are not bound.

The application of the graph pattern in the graph section is different: the graph does not need to be matched against the Target RDF Dataset, but instead to be stored into it. In this case, satisfying the graph equals to satisfying the set of all write operations. A write operation of a graph pattern GP into a graph G succeeds if all the three elements (subject, predicate, object) of the triples in GP are bound (instantiated). Writing of the whole graph pattern does not fail if the subgraph inside an OPTIONAL clause fails to be written (at least one of its triples is not completely instantiated), and the triples of this subgraph are simply left out from the output. If the application of a graph pattern in the graph section fails, no triple is produced for that rule application.

In the example in Figure 2, the placeholder $bookAuthor is never used in the rule, though it is being created inside the nodes section. This brings in another feature of the variable assignment process: whenever a variable in the graph pattern is not bound, before declaring that write to be failed, the processor tries to match it with a placeholder of identical name. In the given example, the variable ?bookAuthor is used two times: the first time as the object of the property bibo:author and the second time as the subject of the property foaf:name. The variable is bound to a match in the target ontology through its presence in the graph pattern of the where section, which looks for already existing authors with the same name. If the triple in the where graph is a match, then the variable is bound and initialized with the URI already assigned to the author with that name. If the match fails, the variable is not bound, but the value of the placeholder with the same name – that is the one initialized by converting the it.uniroma2.Book:author feature – will be used in place. The idea is that, if the author already exists in the target ontology, then its URI is retrieved through the where pattern, otherwise a
URI is generated for it. In other words, the execution of the where section is a solution for the identity resolution problem. Furthermore, the same objective can be achieved by using a converter, especially when sophisticated application or domain specific procedures are needed. The selection among different constructs for the same task should be driven by the user capabilities and needs.

The execution of lazy rules is limited to the initialization of placeholders, since such rules may not define other sections. Moreover, they follow a different invocation strategy, based on bindings declared inside other matched rules.

PEARL offers many other language features (e.g. rule dependencies, UIMA list management, conditional assignments etc...), widely documented in [24].

3.3.3 A Comparison of PEARL with Other Existing Approaches

While discussing background and motivation, we have mentioned existing systems and tools aimed at triplification of data extracted from unstructured content. We analyze here the positioning of PEARL with respect to them. Figure 4 presents a categorization of the main approaches. The first row represents those systems that initially perform a vocabulary agnostic triplification of extracted data, then reduce the generation of vocabulary-specific triples to an RDF refactoring problem. The second row presents a specular approach, in which extracted data are transformed by means of rule languages into an intermediate model, which can be straightforwardly projected as RDF triples. The approach in the third row introduces a single step conversion, which necessarily requires a dedicated language, combining capabilities for querying and manipulating both the source format and RDF.

RDF UIMA CAS Consumer, Clerezza-UIMA integration, and Apache Stanbol all fall in the first category. The advantage in adopting such an approach lies exactly in the early migration to RDF, which allows the reuse of existing languages and technologies (such as SPARQL-CONSTRUCT and SWRL) for projecting source data towards the target vocabulary. On the other side, these languages are not optimized for the specific task, thus implementing the required transformations is not immediate. A lossless serialization of a CAS into RDF clearly depends on some vocabulary for representing [27, 28, 29] the references to (portions of) the subject of analysis.

The second approach can be actually implemented through transducing tools existing for most architectures, such as JAPE (Java Annotation Patterns Engine) [30] for GATE or Ruta [31] (formerly, TextMarker [32]) for UIMA. These tools
allows easy processing and conversion of data in the source format (e.g. a Type System in UIMA), which can then be fed to an intermediate model (in the same format) for which a straightforward projection to RDF is defined once for all. The drawback of this approach lies in the inaccessibility of the Target RDF Dataset. In GATE, they overcame this limitation by introducing an ontology access layer that can be used in the right hand side part of JAPE rules. The combination of JAPE and Ontology Access Layer is an example of the third approach. However, this solution lacks the elegance and conciseness of a dedicated language, as the ontology layer is invoked by means of Java code (as JAPE allows for Java code to be used in the right hand side part of a rule).

PEARL falls too in the third category, though with respect to the above case, provides a dedicated declarative language that combines feature path expressions for querying UIMA data with SPARQL-like expressions for accessing the ontology. The possibility to use SPARQL not only for generating RDF triples, but also for querying the dataset to be populated, allows the refinement of the output according to feedback gathered from already existing – and assessed – triples (see end of previous section, or [24] for reading about further feedback-based possibilities).

Approaches combining different languages to support transformation between their respective domains are not new in literature. In the context of Semantic Web Services, where there is a need for transforming RDF content into XML-based (eXtensible Markup Language) [33] languages for Web Services, the XSPARQL [34] proposal combines SPARQL RDF querying capabilities with the XML manipulation of XQUERY [35]. In the context of data lifting for the Semantic Web, DM (Direct Mapping) [36] allows to view relational databases as RDF datasets, whose structure and vocabulary are derived systematically from the input relational schemas. A companion specification, R2RML (RDB to RDF Mapping Language) [37], allows to personalize the mapping, while leveraging SQL (Structured Query Language) for constructing views over the input databases. These two specifications roughly correspond to the approaches represented in Figure 4, in the first and third row, respectively. Facelets [38] provides a view-declaration language for enterprise web applications by exploiting the extensibility of XML to embed its own tags inside standard XHTML web pages, which can thus be easily shared between web designers and developers. By considering how the above manifestations of this task have been addressed in different areas, we can gather further evidences supporting the design of PEARL.

Meta-model type systems [39] and to a lesser extent the Referent type in the OASIS specification support the use of ontological resources in the analysis metadata. Providing that the focus of these metadata is in deep-semantics, then it should be possible to derive systematically their RDF representation. Nonetheless, RDF refactoring could be still necessary, unless the analysis engine are aware of, or even coupled to, the target dataset and vocabularies.
4. Use Cases and Work conducted in the last 15 months

In this section, we detail the improvements that we brought to the CODA framework, and the dedicated applications that we developed for SemaGrow, from month 19 to month 33. The improvements were brought in light of feedback received by real users, or (more frequently in the case of SemaGrow, as CODA is a development framework and our users were primarily end users) experience gained by adopting it for solving challenges provided by real use case scenarios.

4.1 Improvements to the main framework

The introduction of converters (the main feature added to CODA within its finalization as an open-source project, done in in the context of the SemaGrow project, see sections 3.2 and 3.3 of this same deliverable) has proven valuable in expanding the capabilities of the system. Converters helped in dealing with real case scenarios and with the heterogeneity of requirements that large organizations may establish for their datasets, in terms of content to be acquired, target formats etc...

Also, their distributed architecture, thought with the same principles of linked data in mind, but applied to software provisioning, is absolutely innovative, and in line with similar attempts conducted in other recent EU funded projects, such as LiDER7

For this reason, much of the improvements in CODA have concerned the converters architecture, and on improving their action power and range of applications. Other improvements are related to the range of elements addressable in a PEARL document (the document containing the set of triplification rules for lifting extracted content to RDF), and necessary modifications for better allowing integration of the CODA framework inside other hosting environments.

Specifically, this is the list of improvements brought to CODA with respect to the architecture presented in D3.3.1

- General framework improvements: Unit tests, Context
- PEARL Improvements: addressable objects, regexp
- Converters: Extended range, typechecking/validation, SKOS/SKOSXL dedicated converters, additional parameters
- Integration with other systems: more fine-grained API, context, introduction in Semantic Turkey

In the next sections, we will describe them more in details. All of the described changes are also documented by the comments on the commits to the project, available at: https://bitbucket.org/art-uniroma2/coda/commits/all

4.1.1 General framework improvements

4.1.1.1 Unit tests

We have added a long list of unit tests to the framework. Unit tests have become popular in extreme programming, as they are said to reduce the development time if considered as part of core development itself, that is by developing tests even before the functionalities to be tested have been developed (and practically thus allowing their testing as soon as they are ready). Though we have not applied these brilliant methodologies when initially developing CODA, we decide to benefit from the other great advantage brought by unit testing: maintenance. Whenever a software is being updated, some of its functionalities could get corrupted by means of cascades of changes brought by few modifications at its core.

The purpose of tests is in this case to prove that the system “survived uncorrupted” to a series of core changes and that its functioning is still consistent with the expected behavior.

We thus developed a long series of unit tests for verifying the integrity of the system over a large set of use cases. These unit tests (developed in the period of June/July 2014, not long after the first edition of this document was delivered) have already provided their payback during this last year of development, signaling undesired effects of changes brought to

7 http://www.lider-project.eu/
converters or to the PEARL grammar. Many more unit tests have been developed and added together with the functionalities they should test, in the perfect spirit of extreme programming.

### 4.1.2 PEARL Improvements

#### 4.1.2.1 Addressable objects

It is now possible to use the text covered by an annotation (accessible via the coveredText feature) as a value for a placeholder, by stating coveredText in the feature path.

#### 4.1.2.2 Regexp search

UIMA was lacking, at least until the Ruta [31] backg plugin was developed, a mechanism for searching through annotations. Despite the mentioned Ruta is now available, we found useful to allow for a very simple reg-exp based mechanism, inside PEARL rules, to address specific entries in the UIMA CAS. This has been added in July 2014.

### 4.1.3 Converters

The following improvements have been brought to converters, in order to widen their range of possible applications and their generative power.

- Their input can be a feature structure and not just a string.
- Type checking/validation when being called, to check that the right element is being passed for conversion.
- Added a full range of dedicated SKOS oriented ID generators; see: [http://art.uniroma2.it/coda/documentation/pearl.jsf#converters_overview](http://art.uniroma2.it/coda/documentation/pearl.jsf#converters_overview)
- Added the possibility to foresee converters with additional parameters (i.e. other than the standard input content passed in the Node section of PEARL)
- Added the possibility to defined converters with no standard input: actually they are “producers”, as they do not convert any content from the extracted feature, but are useful for generating content based on other information, such as the context, and/or additional parameters.
- Developed additional converters for xsd:date, xsd:time e xsd:datetime.

### 4.1.4 CODA Integration with other systems

#### 4.1.4.1 Fine-grained API for following the triplification process step-by-step

Host applications featuring CODA, or being entirely based on CODA, might need to have more control over the triplification process, and being able to access and eventually control the flow of information on each of its processing steps. For this reason, we improved the quality of CODA APIs, making more transparent the triplification process and allowing more control through programmatic access.

#### 4.1.4.2 Added the notion of context

A context is a software concept defining a series of (possibly interchangeable and/or extensible) background information that software systems should foresee to handle. Thanks to this, hosting applications such as Semantic Turkey or VocBench, can pass their background information to the CODA, such as the reference to the RDF dataset being managed, or the set of prefix-namespace mappings, and so on.

#### 4.1.4.3 CODA bundled as part of the Semantic Turkey framework: custom ranges

Already introduced in other deliverables, Semantic Turkey [40], or in short, ST, is a RDF management system developed by UNITOV, which is also powering VocBench 2.0 [41], the collaborative thesaurus editing platform adopted by partners in SemaGrow. We already had experience of using CODA to power extensions of ST dedicated to content acquisition.

---

8 [http://www.extremeprogramming.org/rules/unittests.html](http://www.extremeprogramming.org/rules/unittests.html)
However, recently we devised an interesting way of reusing all the expressive power of PEARL (the rule language of CODA) for a standard functionality of ST: custom ranges.

See the dedicated section as a specific use case, later in the deliverable.

4.2 Acquisition from Raw Natural Language Content. Relationship-Extraction.

Basing on preliminary experimentation already conducted on CODA, we have worked on a use case for the acquisition of new knowledge for the FAO AGROVOC dataset. FAO conducted a preliminary analysis of AGROVOC to examine the current state of development of the resource, and reported on the “density” of data population with respect to different relations and/or concept branches. It emerged that a few relation were considered by the users to be important, however, and had thus been added to the vocabulary behind AgroVoc, however few or no relationship was instantiated on these relations. In our application of CODA, we have built dedicated extractors for a few of those scarcely populated relations, such as:

- agro:hasPest: a relation between two organisms, in which the second causes harm to the first.
- agro:hasHerbicide: a relation between a weed and a pesticide used to remove it from other plants
- agro:hasInsecticide: a relation between an insect and a pesticide used to remove it from plants it often consumes

acquiring information from unstructured content and possibly by comparing the results with existing datasets covering the same domain (e.g. NALT: [http://agclass.nal.usda.gov/](http://agclass.nal.usda.gov/)), all through an automatic workflow defined in CODA.
An experimentation has been performed with discrete results: a set of 526 triples\(^9\) has been automatically acquired from a corpus of documents coming from Wikipedia, from the Department of Agriculture and Fisheries of the Queensland Government\(^10\), from [http://www.domyownpestcontrol.com](http://www.domyownpestcontrol.com)\(^9\) and other relevant sites.

The extracted triples have been reported into an Excel file containing human readable information (e.g. the preferred labels of the involved concepts), the URIs of the relations and involved concepts and the URLs where the extracted relationships have been observed (to facilitate the validation work for humans). A combo box with possible validation outcomes completed the Report & Validation console of the file.

The file has then been sent to FAO for validation, which split it into two files, one related to the hasPest relation, and the other to the two relations covering the use of Pesticides. The second one is still, at the time of writing, under validation, while the hasPest validation (larger one) has reported that 23% out of all the extracted triples have been considered eligible for being added to the AgroVoc database. The main reason for the rejection of the other triples, is the presence of too generic terms, such as crop, grass and fruit. These relations are actually not wrong, and by checking on the source documents, it was possible to verify that they contained sentences such as:

\[\text{"Delia antiqua, commonly known as the onion fly, is a cosmopolitan pest of crops"}\]\(^11\)

to which, the best that the system could do, is to report what was found on the (reputable) page. However, in some cases the experts simply preferred to avoid too generic relations and assert only those very specific ones (such as which specific insects of the diptera family feast on which specific kind of crop). This is the reason for which we reported the 23% as a number of triples eligible for publication, and not as the number of correct triples (which would be many more). Also, a combination of this system with some simple tagging of concepts in a pre-processing phase, in which users mark which concepts they would prefer to avoid even though they are mentioned in the text, would make the percentage much closer to the expected output.

\(^9\) The previous deliverable reported \(~2000.\) Actually this number has been automatically reduced (thus with no human intervention) in order to remove redundant triples (e.g. trivially, duplicates, inverse relationships, etc..), thus reducing the number to 631

\(^10\) [https://www.daf.qld.gov.au](https://www.daf.qld.gov.au)

The aim of this experimentation is however not to cover the specific use case (the results of which depend on the quality of the specific system built on top of CODA), but to exploit the lessons learned through it, in order to make the CODA framework even more powerful and flexible in dealing with progressively broader situations and scenarios.

Future work following the project will possibly include a more general integration (i.e. not a dedicated system such as for AgroIE) of CODA automatic workflows into Knowledge Management platforms, for interactively supervised knowledge acquisition, or validation of automatically acquired information. We are examining the platforms to be chosen for this integration. The most prominent candidate is VocBench, which has already been chosen as the end-user platform for hosting the alignment components developed in T3.2 and is seeing in these months a sharp diffusion among several organizations, or Semantic Turkey [40], the desktop counterpart of VocBench [42]. Mixed approaches are also possible, with local desktop tools carrying the acquisition process (such as in Figure 5) and their results being submitted to web platforms for collaborative validation and consensus achievement.

4.3 Acquisition from Semi-structured content. Spreadsheet processing.

Another use case, which actually covers automatically lifting semi-structured data to structured RDF datasets is the import from spreadsheets. Spreadsheet files are, given their intrinsically structured form and yet easy-to-use capabilities, the preferred exchange formats for people working with thesauri or large datasets and with no specific background in Knowledge Organization & Management. The often-recurring request that any vendor/developer of RDF Knowledge Management systems gets is: “why not enabling Excel-import? That would ease things a lot!”. The problem in this statement is that it is often made by people who, by lacking the background in the field, do not know that Excel (just to cite the most popular spreadsheet format, and the one which is mostly cited) is considered only semi-structured exactly because of the lack of any semantics and of any specific structure organization. In many cases, there are a number of assumptions which are only in the mind of the person creating the spreadsheet, and which require even long discussion to understand the nature of the content, and finally the capacities (and the effort) necessary for developing specific import routines.
When we initially tried to address the problem, using CODA seemed like a forced and out-of-scope approach, as it seemed to be a yet too powerful system for addressing a very narrow case. The question was if it was possible to find a much more easy-to-adopt solution considering the specific case. Then, the more and more we worked on it, the more we felt the need for the powerful features of PEARL (for both RDF resources and graph generation). In addition, the few assumptions for the format of the spreadsheet file, recurring in almost all cases, could be modelled in the form of a dedicated UIMA processor, which would thus not need to be re-implemented case by case. One typical assumption is that people uses spreadsheet files on a row-per-resource basis (otherwise the spreadsheet becomes too complex for them).

In addition, we thought that in a convention-over-configuration approach, many patterns recurring quite often in input spreadsheets (though not so frequently to be considered “assumptions”) could be recognized and translated into specific PEARL transformation patterns. Also, presence in the spreadsheets headers of explicit references to existing vocabularies (e.g. using a qname or full URI of a property of a RDF vocabulary, instead of a generic header) could ease again the generation of a PEARL skeleton by guessing the property to be used (and, in some case, even more complex graph structures than mere single triples obtained from row-columnheader-cellvalue combos). In any case, producing automatically a PEARL document (and basing on an already built UIMA component and a dedicated Type System for spreadsheet processing), gives then the full power to apply any modification needed to cover specific exigencies, case by case.

In Figure 7, a user-friendly solution that integrates the spreadsheet-import system into the ontology editing system Semantic-Turkey is shown.

The system described until now has been released for the first time on the 27th of October 2014, with project name: Sheet2RDF: \http://art.uniroma2.it/sheet2rdf/\.

Sheet2RDF distribution is threefold: it can be used as a platform integrable set of OSGi services, as a command line tool, or as an extension for Semantic Turkey, benefiting from its UI. A porting for the collaborative platform VocBench is also foreseen (as it will mostly require UI development, being VocBench based on Semantic Turkey for RDF management and being able to reuse its extensions as well), though the typical use-scenario for Sheet2RDF is more compatible with the advanced users foreseen by Semantic Turkey than on the controlled collaborative environment of VocBench. Also, the Firefox interface of Semantic Turkey (and thus the Sheet2RDF extension) can be seamlessly used – through a privileged access to the same Semantic Turkey server instance – to modify the content of VB managed projects, thus making the system virtually already available to the range of VB adopters.

By considering current use-cases, such as the continuous flow of information coming to AGRIS from heterogeneous data providers from all over the world for incrementing its content, other interesting features came to our mind in the form of interesting CODA improvements (or extensions/applications).

### 4.3.1 Improvements

From its first release in October 2014 until now, several improvements have been brought to the system, following related improvements in CODA, and focusing on the human-computer interaction possibilities offered by Semantic Turkey, and again, as for the other systems, working on the lessons learned from active use of the tool inside and outside of SemaGrow.

#### 4.3.1.1 UI Improvements

Colored headers in the spreadsheet resume inside Sheet2RDF on ST/Firefox provide information about which headers of the spreadsheet have been mapped and which ones are still not.

Headers have been initially decorated with a context-menu (later an entirely dedicated dialog) which allows to visually define the mapping to the target ontology/thesaurus. This allows for a certain expressive power, whereas more complex mappings need still to be defined through PEARL, benefiting from its full expressive power and transformation capabilities. However, most of the trivial mappings can be guessed by the system automatically, or can be defined by the user through this UI wizard, header per header.
4.3.1.2 Business logic Improvements

The UI wizard can now rely on an automated search on the target ontology, in order to simplify the property selection task of the user. We added a configuration menu, in order to store several user preferences on each user client.

Input/output: We added support for all known triple serialization formats, and allowed to import Open Office spreadsheets other than Microsoft Excel ones.

4.3.2 Adoption of the system

A number of users, whom we are not able to track down if not through their inquiries on the Semantic Turkey mailing list, has already adopted the system. The system has then been also adopted in a project, AgInfra, which has been explicitly said to benefit from scientific/technological outcomes of SemaGrow. The following thesauri developed (or made available) in the context of AgInfra and related efforts have been ported to RDF thanks to the use of sheet2RDF:

- AgInfra_SoilDB:
- AgInfra_Types:
- 4 COAR projects: Classification Schemes, Date Types, Resource Types, Resource Versions

4.3.2.1 AgInfra thesauri ported to RDF

A few KOSs were created during the agINFRA project by partners of the consortium, in order to address gaps that had been identified. The need to expose as LOD a local set of key terms concerning soil used by a project partner: publishing these concepts as LOD allowed to map them explicitly to the corresponding USDA concepts and INSPIRE terms in the INSPIRE Registry whenever possible; this led to the extraction, triplification and exposure of the CRA Soil Terms: http://vocabularies.aginfra.eu/cra-soil-terms;

The need to publish as LOD the values used in AGRIS for “resource type” so that they can be used by data providers as URIs and that they can be mapped to other standard values (like the Dublin Core Types) whenever possible; this led to the publication of the agINFRA Resource Types vocabulary: http://vocabularies.aginfra.eu/aginfra-resource-types;

The following table provides information about the two vocabularies and their hosting environments.

<table>
<thead>
<tr>
<th>Maintenance Tool: VocBench</th>
<th><a href="http://artemide.art.uniroma2.it/vocbench2/">http://artemide.art.uniroma2.it/vocbench2/</a></th>
</tr>
</thead>
<tbody>
<tr>
<td>Platform Hosting:</td>
<td>University of Rome, Tor Vergata</td>
</tr>
<tr>
<td>Data Maintainer:</td>
<td>Food and Agriculture Organization of the United Nations</td>
</tr>
<tr>
<td>KOSs managed here in the context of AgInfra:</td>
<td></td>
</tr>
<tr>
<td>CRA Soil Terms</td>
<td></td>
</tr>
<tr>
<td>agINFRA Resource Types</td>
<td></td>
</tr>
<tr>
<td>Maintenance:</td>
<td>Food and Agriculture Organization of the United Nations</td>
</tr>
<tr>
<td>Namespaces</td>
<td></td>
</tr>
<tr>
<td>Soil Terms:</td>
<td><a href="http://soilmaps.entecra.it/kos/">http://soilmaps.entecra.it/kos/</a></td>
</tr>
<tr>
<td>Maintainer:</td>
<td>Consiglio per la Ricerca in Agricoltura (CRA)</td>
</tr>
<tr>
<td>Maintenance:</td>
<td>Food and Agriculture Organization of the United Nations</td>
</tr>
<tr>
<td>RDF dump</td>
<td></td>
</tr>
<tr>
<td>Soil Terms:</td>
<td><a href="http://202.4139.84:10035/catalogs/fao/repositories/agINFRA#query/d/SELECT%20%25s%20?%20%WHERE%20%7B%20%skos:inScheme%20%3Chttp://soilmaps.entecra.it/kos%3E%20%7D">http://202.4139.84:10035/catalogs/fao/repositories/agINFRA#query/d/SELECT%20%s%20?%20%WHERE%20%7B%20%skos:inScheme%20%3Chttp://soilmaps.entecra.it/kos%3E%20%7D</a></td>
</tr>
</tbody>
</table>
4.3.2.2 The CRA Soil Terms KOS

Consiglio per la Ricerca in Agricoltura (CRA), a partner in the AgInfra project, contributed its SoilDB thesaurus\(^{12}\) to the project, in the effort to bring it to the RDF world and continue to develop it as a SKOS thesaurus.

The original database was very well structured and the terms used in the various classifications (from soil types to the allowed values for several physical and chemical properties) were all stored in a table, with hierarchical relations between them. For most of these terms, the corresponding literal values in international standard classifications were stored in a table field.

The database has been ported to Excel (Error! Reference source not found.) through an exporter available from their legacy information management system, and thus fed to Sheet2RDF.

The work done by the CRA Research Centre for Agrobiology and Pedology (CRA-ABP) in mapping most of the terms to the corresponding URIs of concepts in other international classifications that have been published as LOD allowed to include the mapping to external URIs produced for the final SKOS version, thus making the KOS really Linked Data.

The international classifications to which the local values in the CRA classifications were mapped are:

- the USDA Soil Taxonomy (Soil Survey Staff, 2014), also part of the US National Agricultural library thesaurus (NALT);
- the INSPIRE "Registry" (European Commission Joint Research Centre, 2013), a public reference directory of all the published "registers" (in INSPIRE terminology) assigning published identifiers to specific controlled values;
- the Multilingual Soil Thesaurus (SoilThes), an extension of the General Multilingual Environmental Thesaurus (GEMET) developed in the eContentplus project GS SOIL.

\(^{12}\) http://soilmaps.entecria.it/
Some of the above classifications have been published as Linked Data, therefore the publication of the CRA classifications included the mapping to the URIs of the corresponding concepts in the above vocabularies. Some occasional mappings to AGROVOC are included as well; crosswalks to AGROVOC are as well possible through the mapping between the NALT (of which the Soil Taxonomy is a branch) and AGROVOC.

In Figure 9 it is possible to observe the output of the transformation, in the concept tree view of the VocBench platform, which is being currently adopted to maintain the vocabulary.

Figure 9. Visual representation of an extract of the CRA Soil Terms vocabulary in the VocBench platform: sub-tree under the concept “Soil Body”
4.3.2.3 The agINFRA Resource Types KOS

Another type of controlled values that underwent standardization and publication as LOD in the AgInfra project was a controlled list of values (with partial hierarchy) used to define resource types (for example for the dc:type property used in most bibliographic records).

The objective here was not so much that of providing an authoritative KOS (some already exist, like the Dublin Core Types, but are not exhaustive) as that of assigning URIs to these values used locally and map them whenever possible to URIs used in other more authoritative KOSs (like the aforementioned Dublin Core Types: http://purl.org/dc/dcmitype/).

The AgInfra consortium partners involved in the effort extracted literal values from the AGRIS bibliographic database (aggregating records from 150 data sources, thus representing a huge variety in the use of the dc:type property) and published them as LOD linking them to existing external URIs whenever possible. The description of these URIs was again provided as an Excel file, which was later submitted to UNITOV, which in turn realized a conversion through Sheet2RDF.

4.3.2.4 The four COAR projects

The mission of the Confederation of Open Access Repositories (COAR) is to facilitate the greater visibility and use of research outputs through a global network of open access repositories. To that end, COAR actively works towards the alignment of repository networks and other systems via technical and semantic interoperability, as well as common approaches to policy and services.

In 2014, COAR established the Interest Group “Controlled Vocabularies for Repository Assets” with the objective of developing a set of controlled vocabularies for the bibliographic metadata elements used in records describing research outputs. Controlled vocabularies, if applied consistently, offer a uniform standard terminology for describing content, and assist with resource discovery, sharing and reuse of metadata and content, as well as integration between different systems.

---

13 https://www.coar-repositories.org/about/coar-ev/strategic-plan/
14 https://www.coar-repositories.org/activities/repository-interoperability/ig-controlled-vocabularies-for-repository-assets/
15 Following the recommendations from “How to select appropriate encoding strategies for producing Linked Open Data (LOD)-enabled bibliographic data” http://aims.fao.org/lode/bd
This Interest Group complements the work of a number of other COAR initiatives, including the COAR-CASRAI Working Group on Interoperability of Open Access Networks\textsuperscript{16}, the COAR Interoperability Roadmap\textsuperscript{17}, and the Strategic Committee for Aligning Repository Networks\textsuperscript{18}. Ultimately, the aim is to share the controlled vocabularies developed through this Interest Group, with these other groups, so they can be eventually adopted by organizations and initiatives in the broader environment. Inevitably, the process of adoption will be an iterative one, whereby relevant vocabulary elements are shared once stabilized, and eventually adopted by other groups who are working on specific interoperability solutions in a given area.

As a result of this work, we distinguish into the COAR Set of Controlled Vocabularies governed and maintained by the authority group and other vocabularies and dictionaries that are governed by external authorities.

Initially the COAR Set of Controlled Vocabularies consists of:

- vocabulary about resource type to identify the genre of a research resource
- vocabulary about access mode to declare the degree of ‘openness’ of a resource
- vocabulary about version to express a specific version of a resource
- vocabulary about date type to tag an event in the lifecycle of a resource
- vocabulary about classification schemes

In addition the set is complemented by recommended vocabularies and authority files governed by external authorities:

- vocabulary about licenses governed by creative commons\textsuperscript{19}, and governed by the Open Data Institute\textsuperscript{20} in order to express the legal level of re-use and openness\textsuperscript{21} and reflect requirements from Open Access Policies from Funders\textsuperscript{22};
- identifier vocabulary governed by the Library of Congress\textsuperscript{23};
- fundRef\textsuperscript{24} that provides an authoritative list of funders and thus can be used to make relations between the research output related to a funded project;
- CRediT is working on an open standard for expressing roles intrinsic to research, http://credit.casrai.org/proposed-taxonomy/, this is a piece of information increasingly codified by journals and may pave the way for a new metadata in repositories.

All of these vocabularies have been submitted as excel files, have been ported to SKOS-XL by use of Sheet2RDF, and are currently being maintained through an instance of VocBench available at: http://artemide.info.uniroma2.it/vocbench2

4.4 Integration of CODA inside Semantic Turkey: Custom Ranges

One very important aspect in adapting to vocabularies and/or evolution of the ones being adopted for the datasets being managed, is the capability of data/model maintenance systems to adapt to the characteristics of these vocabulary, and of the particular modeling patterns that some of them may exhibit, without requiring to build/develop dedicated solutions.

\textsuperscript{16} [URL]
\textsuperscript{17} [URL]
\textsuperscript{18} [URL]
\textsuperscript{19} vocabulary described in RDF: https://creativecommons.org/ns and registered in LOV: http://lov.okfn.org/dataset/lov/details/vocabulary_cc.html
\textsuperscript{20} [URL]
\textsuperscript{21} as addressed in the NISO recommendations, http://www.niso.org/workrooms/ali/
\textsuperscript{23} one such approach is: http://www.loc.gov/standards/sourcelist/standard-identifier.html
\textsuperscript{24} http://www.crossref.org/fundref/ and the fundref schema: http://www.crossref.org/schemas/fundref.xsd
Already introduced in other deliverables, Semantic Turkey [40], or in short, ST, is a RDF management system developed by UNITOV, which is also powering VocBench 2.0 [41], the collaborative thesaurus editing platform adopted by partners in SemaGrow. We already had experience of using CODA to power extensions of ST dedicated to content acquisition (the screenshot in section 4.2 represents a first attempt at integrating CODA with ST, for the specific application AgroIE). However, recently we devised an interesting way of reusing all the expressive power of PEARL (the rule language of CODA) for a standard functionality of ST: custom ranges.

Custom Ranges are specific complex objects the design of which is specified through a graph pattern. The idea is to use the same knowledge acquisition mechanism of CODA in order to populate forms for creating complex objects to be used as targets of a property (instead of a single resource, be it a literal or URI). This will help users in representing complex objects through user-friendly forms: these forms can be populated with human understandable information, which will then be composed and normalized by the PEARL rule.

In order to support this functionality, we have built a CustomRangeProvider class that is invoked internally by the standard service for getting the ranges of properties in Semantic Turkey.

This is an example of the return type for the range of a property:

```xml
<stresponse request="getRange" type="reply">
  <reply status="ok"/>
  <data>
    <ranges rngType="resource">
      <uri explicit="false" role="cls">http://www.w3.org/2000/01/rdf-schema#Resource</uri>
    </ranges>
  </data>
</stresponse>
```
A further attribute: "custom" (which may be set to true/false) has been added to inform the client that a custom range is also available.

If the value of the attribute is true, then we will have, as sibling of the <ranges> element, a further element, called <customRange>, which contains the serialization of the customRangeEntries (up to you how to make it, but there is more info down below).

The rationale behind custom ranges is that creating complex objects is not a trivial operation in a standard ontology editor. Even the most simple ones, such as skos:Labels, which require the following pattern to be realized:

```
:aConcept skosxl:prefLabel :aLabel
:aLabel skosxl:literalform "thelabelliteral@<langtag>"
```

or reified skos definitions:

```
:aConcept skos:definition :aDef
:aDef rdf:value "the literal part of the definition@<langtag>"
```

require – if not properly handled – users to first create the uri for the skosxl:Label :aLabel, and then add its literal form. Obviously, this is not viable: such a procedure would puzzle the mind of most non-RDF-savvy users, and in any case it would be really an uncomfortable procedure for any user, to say the least.

In VocBench, currently (version 2.3), the management of skosxl:Labels and reified SKOS definitions is hard-coded in the application: the user simply is prompted with a request for the literal form of the label/definition, and the application takes care of generating the URI in the middle and of attaching eventual metadata (date of creation/update) to it. Then, there are then options for exporting the data by keeping reified SKOS definitions and SKOS-XL Labels as they are, or for flattening them upon export.

However, beyond the general SKOS/SKOS-XL model options, specific users could have plenty of necessities where they need dedicated forms for creating arbitrary domain objects (e.g. described by their domain ontology) represented by a nested subgraph and not by a single RDF entity. Custom ranges allow covering such a necessity. Where the definition contained in an ontology is not enough to represent such necessities, the administrator of a project can represent them as PEARL acquisition patterns.

The example in Figure 12 shows the use of a PEARL rule for a classic SKOS reified definition. The userPrompt is a special feature structure adopted in this custom use of CODA: instead of being pre-existing, and generated by extraction of

```
rule it.uniroma2.art.semanticturkey.reifiedDefinition {
  nodes = {
    reifDefId  uri(st:rndURI) .
    defLit     plainLiteral  userPrompt/definition
    source     literal^^xsd:String userPrompt/source
    date       literal^^xsd:DateTime(st:currentDatetime) .
  }

  graph = {
    $reifDefId  rdf:value $defLit
    $reifDefId  agro:source $source
    $reifDefId  dct:created $date
  }
}
```

**Figure 12.** PEARL rule used in ST for driving the form generation for the acquisition of a reified skos:definition
content from some unstructured source, the userPrompt is actually a driver for generating the form. So, the system will parse the PEARL file by first in order to:

- Generate the userPrompt feature structure
- Get the structure of the form

and will then store the content provided by the user on the form, in that feature structure. The rest will proceed as in CODA, the content will be processed by means of converters and transformed into RDF entities. Triples will be then generated on the basis of the rest of the PEARL file (mainly the GRAPH section).

Each custom range is defined in ST by the following elements:

<table>
<thead>
<tr>
<th>field</th>
<th>Explanation</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>a namespaced string</td>
<td>it.uniroma2.art.semanticTurkey.reifiedNotes</td>
</tr>
<tr>
<td>name</td>
<td>a short name which at the same time may suggest the nature of the range (it is also be used in the UI to present the entry as a choice)</td>
<td>reified note</td>
</tr>
<tr>
<td>description</td>
<td>a natural language understandable explanation of the range type (a tooltip for when the mouse hovers over the choice)</td>
<td>&quot;a reified note, described by an RDF Resource linked through rdf:value to its lexical representation and to other metadata&quot;</td>
</tr>
</tbody>
</table>
there can be various types to fill a custom entry, for now, just node/graph. The value here determines the existence of another homonymous field

<table>
<thead>
<tr>
<th>type</th>
<th>node, or graph</th>
</tr>
</thead>
<tbody>
<tr>
<td>ref (in case of node as value of type)</td>
<td>a CODA converter</td>
</tr>
<tr>
<td>ref (in case of graph as value of type)</td>
<td>a PEARL rule which has been loaded from the set of rules for generating forms.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>type</th>
<th>node, or graph</th>
</tr>
</thead>
<tbody>
<tr>
<td>ref (in case of node as value of type)</td>
<td>a CODA converter</td>
</tr>
<tr>
<td>ref (in case of graph as value of type)</td>
<td>a PEARL rule which has been loaded from the set of rules for generating forms.</td>
</tr>
</tbody>
</table>

Figure 14. the generated form for entering values for a reified skos definition
The custom-ranges system has been already developed and finalized and it will be featured by the next version of Semantic Turkey to be released (0.12). Figure 11, Figure 13, Figure 14, Figure 15.

UNITOV foresees to introduce Custom Ranges in VocBench as well, though the process will not be immediate due to the less flexible layers of VocBench associated to the controlled editing of resources.

![Resource View](image)

**Figure 15.** Visualization of the reified SKOS definition in the editor panel of ST
5. REFERENCES


