Int J Digit Libr (2009) 10:179–199
DOI 10.1007/s00799-010-0059-z

Semantic Web enabled digital repositories

Dimitrios A. Koutsomitropoulos · Georgia D. Solomou · Andreas D. Alexopoulos · Theodore S. Papatheodorou

Published online: 19 June 2010

Abstract Digital repositories and digital libraries are today among the most common tools for managing and disseminating digital object collections of cultural, educational, and other kinds of content over the Web. However, it is often the case that descriptive information about these assets, known as metadata, are usually semi-structured from a semantics point of view; implicit knowledge about this content may exist that cannot always be represented in metadata implementations and thus is not always discoverable. To this end, in this article we propose a method and a practical implementation that could allow traditional metadata-intensive repositories to benefit from Semantic Web ideas and techniques. In particular, we show how, starting with a semi-structured knowledge model (like the one offered by DSpace), we can end up with inference-based knowledge discovery, retrieval, and navigation among the repository contents. Our methodology and results are applied on the University of Patras institutional repository. The resulting prototype is also available as a plug-in, although it can fit, in principle, any other kind of digital repository.

Keywords Semantic Web · Metadata · Interoperability · Digital repositories

D. A. Koutsomitropoulos (✉) · G. D. Solomou · A. D. Alexopoulos · T. S. Papatheodorou
High Performance Information Systems Laboratory, Computer Engineering and Informatics Department, School of Engineering, University of Patras, Building B, 26500 Patras-Rio, Greece
e-mail: kotsomit@hpclab.ceid.upatras.gr

G. D. Solomou
e-mail: solomou@hpclab.ceid.upatras.gr

A. D. Alexopoulos
e-mail: alexopoulo@hpclab.ceid.upatras.gr

T. S. Papatheodorou
e-mail: tsp@hpclab.ceid.upatras.gr

1 Introduction

At the current verge of the Semantic Web deployment incentive, it is crucial to see how it can fit into and influence well-known and established information management paradigms. Such an investigation is substantial, not only because Semantic Web technologies have become "hype" in IT trends, but also for it allows us to find novel improvements and added value to common chores, such as resource discovery, metadata manipulation, and interoperability.

The Semantic Web infrastructure relies, among others, on specifications for expressing ontologies in web-compatible format (OWL [21]) and lately, on programming efforts for manipulating such ontologies, like the OWL API [9] and the Protégé 4.0 code-base (the CO-ODE project).[1] To be able to fully reap the benefits of a *Semantic* Web, reasoning over ontological information is of exceeding importance, a fact that was sometimes overlooked in the past, possibly because of the immaturity of available tools and techniques. To our belief, the ability to infer implied information over declared facts and assertions is one of the most prominent reasons to investigate and implement the Semantic Web, in a sense that adds an "AI" flavor to the current Web [8].

Therefore, in this article we present and document a process that builds upon the well-known *digital repositories* paradigm and enhances it with the Semantic Web's features, an idea earlier envisaged in [14]. In other words, the main goal that drives our efforts is not to re-implement a digital repository system using Semantic Web APIs and technologies, but to provide inference-based knowledge discovery, retrieval, and navigation *on top* of such a system, based on existing metadata and other semi-structured information.

[1] http://www.co-ode.org/.

The merit of such an approach is multiple:

- First, we show how the Semantic Web can really contribute to information management practices by enabling features that would otherwise be impossible, such as semantic querying and navigation.
- We proceed in a way that not only leaves the underlying system architecture intact, but also maintains the internal data model used by the system. In other words, our enhancements build "on top" of the existing programmatic components and integrate with them in an interoperable way.
- Leaving the data model intact, our approach reuses existing metadata that are automatically manipulated, in order to extract added value and implicit semantic relations among them. Therefore, a solution to the Semantic Web bootstrapping problem is suggested [4], since we are able to auto-produce meaningful, ontological descriptions of resources from existing metadata. At the same time these new ontological descriptions are self-contained (as ontologies are) and can be directly consumed by other Semantic Web enabled applications, achieving thus a *semantic* level of interoperability.
- Being independent of the system architecture and relying mostly on interoperability interfaces, our approach can be likewise applied to enhance any modern Digital Repository or Digital Asset Management (DAM) system.

To prove our concept, we describe a concrete, working prototype that provides for inference-based search and navigation on top of the DSpace digital repository system.[2] DSpace has become a popular open-source digital repository solution with one of the most rapidly growing user bases worldwide. DSpace metadata follow the Dublin Core (DC) specification by default, while it is possible to import and use other metadata schemata as well. Our work and results are based on real-world data and applied to the official University of Patras institutional repository that is based on DSpace.[3] Its metadata are based on the original DSpace schema extended with learning object metadata (LOM).

We also show the benefits gained from such an approach, mainly by presenting some representative examples and commenting on the obtained results. This evaluation process considers the quantitative and qualitative characteristics of our ontologies and tries to capture the behavior of the implemented prototype when posing semantic queries.

This article is further organized as follows: First an overview of related work on digital libraries and semantics is given; then, we present the construction of the repository's ontology, following the *semantic profiling* technique [15] and explain the automatic population procedure. Further, we document the implementation of the ontology management, search, navigation, and reasoning services and how they can wrap around the DSpace system. Finally, we give specific examples and results that demonstrate these new capabilities and summarize the conclusions of our work.

The impact of this work on the DSpace system, known as *Semantic Search for DSpace*, is now available as a plug-in, officially acknowledged by the DSpace community.[4] However, this impact is not limited to DSpace only, since our implemented features do not *of necessity* bind to the underlying system in an architectural sense, and can be likewise applied to every sort of repository with metadata exportable and available in machine-readable formats.

2 Related work

Some widely adopted mechanisms for digital repositories that, similar to our work, appear to utilize Semantic Web technologies, like RDF and ontologies, are BRICKS, SIMILE, Fedora and JeromeDL plus the newer Talia.

The aim of the BRICKS project [27] is to design, develop, and maintain an open user- and service-oriented infrastructure to share knowledge and resources in the cultural heritage domain. It provides a set of useful services on top of which each content provider can develop his own application and user interface. Similarly, FEDORA [6] is an interoperable, distributed repository service that can serve as a fundamental component in an open digital library infrastructure. Both BRICKS and Fedora provide the basic architecture upon which digital library applications can be developed and deployed. Their functionality is further enhanced by supporting some basic—albeit not so powerful—Semantic Web technologies, like the expression of relations between objects in RDF and the retrieval of data through the evaluation of queries using SPARQL or other RDF query languages.

SIMILE[5] is a research project focused on enhancing interoperability among digital assets, schemata, ontologies, metadata, and services. It is not a full-fledged digital library system, but provides a set of tools for discovering and navigating digital resources on the Web. It was originally motivated by DSpace in order to leverage and extend its facilities. Through the application of RDF and Semantic Web techniques, SIMILE offers DSpace improved support for arbitrary schemata and metadata and provides an architecture for disseminating digital assets. Among SIMILE's implemented tools, the one that pertains mostly to our work is a faceted browser, known as Longwell (its DSpace version is called Dwell), which combines the ease of unstructured, 'Google-like' free text search, with the ability to cross-section the data along the dimensions

[2] http://www.dspace.org/.

[3] http://repository.upatras.gr/dspace/.

[4] http://www.dspace.org/add-ons-and-extensions/#semantic.

[5] http://simile.mit.edu/.

of structured metadata. Furthermore, SIMILE provides tools to merge lexical or semantic variants via simple inferencing. Despite all these, current implementation of SIMILE seems to lack the ability to offer reasoning-based querying.

Another digital repository system related to our work is JeromeDL [18], a *Social Semantic Digital Library*. It makes use of Semantic Web and Social Networking technologies and its aim is to enhance both interoperability and usability facilities in the domain of digital libraries.

JeromeDL's metadata are stored in RDF all along, using MarcOnt as a mediating ontology and managed by a corresponding RDF store (Sesame). These metadata can then be retrieved through a direct RDF query service (RDQL, SeRQL, and SRQL) or by adopting a simple and more appealing solution to the non-expert users, based on natural language query templates. A level of inference is supported through a simple recommendation engine based on Prolog. JeromeDL seems to solve the "semantic bootstrapping" problem following a bottom-up approach, where the ontological schema is constructed and populated in advance; in such a way the problem of retrieving semantic implications and inference-based results also from flatly organized relational data base sources is circumvented.

Nonetheless, JeromeDL's strong focus on social services and the (semantic) tagging of resources definitely constitute two important features of the platform's underlying technology that justify its sound presence in the domain of Social Semantic Digital Libraries.

Finally, an infrastructure somewhat similar to JeromeDL and to our work is the newly appeared library platform Talia [24], being developed with the Ruby on Rails Web framework. Talia keeps its metadata into a relational DB Schema and keeps it "in sync" with an RDF data store (Redland). In addition it provides a unified query interface for both database and RDF metadata using SQL or SPARQL, as required. However, Talia does not do any inferencing on the RDF data and leaves this responsibility to the underlying RDF store.

In the following Table 1 we identify and summarize the Semantic Web features available in these systems. We also review them according to their ability to perform semantic queries and inferencing over their content.

3 Creation and population of the ontological model

In this section we give an account of how we have developed a Semantic Web ontology out of the repository's metadata, based on which we can employ our complementary, semantics-aware services. First, we give an overview of the methodology followed and the reasoning behind it that can serve as a paradigm for constructing rich ontologies out of semi-structured sources, which is often the case in digital repositories. Then, we document in some detail the steps that actually implement our ontology, roughly following the semantic profiling technique. Finally, we explain the process for the automatic creation and population of the ontological instances from the repository's raw metadata.

3.1 Overview

In many standard repository configurations (including the DSpace digital repository software), resources are described

Table 1 Semantic characteristics of some popular digital repository infrastructures

	Querying capability	Semantic Web features
BRICKS	Ontology-based search (SPARQL) Inference support	Based on RDF and OWL for handling heterogeneous metadata internally (but RDF and OWL are completely hidden from end users and high level applications and they are only exposed to machines)
FEDORA	RDF query (SPARQL)	Expression of relationships among digital resources
	Inference support	Contextualization
SIMILE	Free text search (facet browsing)	Cross-section of data along the dimensions of structured metadata (mapping of heterogeneous metadata in RDF)
		Merging of lexical or semantic variants via simple inferencing
JeromeDL	RDF query (RDQL, SeRQL)	Description of content with respect to a JeromeDL-specific ontology
	Natural language queries	Ontologies feedback service
	Basic inference capabilities (based on Prolog)	Identity management
		Semantic query expansion
		Extensive conceptualization
Talia	RDF query (SPARQL)	RDF based (built upon Active RDF)[a]
	Inferencing is left to the underlying RDF store	Use of ontology descriptions

[a] http://www.activerdf.org/

Fig. 1 Detailed item view in DSpace

DC Field	Value	Language
dc.contributor.author	Κουτσομητρόπουλος, Δημήτριος	el
dc.contributor.author	Koutsomitropoulos, Dimitrios	en
dc.date.accessioned	2006-12-15T13:13:49Z	-
dc.date.available	2006-12-15T13:13:49Z	-
dc.date.issued	2006-12-15T13:13:49Z	-
dc.identifier.uri	http://hdl.handle.net/1987/104	-
dc.description	HPCLab	en
dc.description	Εργαστήριο Πληροφοριακών Συστημάτων Υψηλών Επιδόσεων	el
dc.description.abstract	Στην παρούσα παρουσίαση περιγράφονται ενέργειες προετοιμασίας πρωτοτύπων που στόχο έχουν τη διατήρηση και τη διάσωση της ποιότητας του περιεχομένου.Παράμετροι που αναφέρονται και αναλύονται είναι η καταλληλότητα του εξοπλισμού και των συνθηκών περιβάλλοντος και η μεταχείριση πρωτοτύπων.	el
dc.description.abstract	The presentation above describes preparation actions that intent to the preservation and the rescue of the content quality.Factors like equipment and environment appropriateness as well as prototype usage are analytically adverted.	en

based on the DC Metadata Element Set (DCMES) which is often implemented as a flat aggregation of elements. This is also true for qualifiers, which are not always implemented as sub-properties of main elements; rather, they often appear at the same level as parent elements and the sub-element/qualifier relationship is maintained only in the label. Although not currently acceptable by DCMI, this practice is adopted by lots of downstream systems, including DSpace. For example a sample metadata record in the DSpace-based University of Patras institutional repository looks like the one in Fig. 1.

The semantic interpretation of the DC model, which, as we see, is not always represented in applications, is formalized through the DCMI Abstract Model (DCAM) specification [26] as well as the most recent recommendation for expressing DC in RDF [23]. These documents virtually suggest an ontology of DC, expressed in RDF(S), a Semantic Web standard. Such a DC ontology bears its own semantic structure that may be taken advantage of in order to enable more refined descriptions of resources.

Similarly, the AIFB institute at the University of Karlsruhe maintains an implementation of the DC, this time in OWL format.[6] In it, an incremental approach is followed where finally the full underlying semantics of DC are accommodated, rendering unfortunately the ontology undecidable (since it is in OWL Full).

Protégé also fosters a DC implementation in OWL,[7] where all properties are declared simply as annotation properties. Additionally, DRC Inc. proposes an OWL DC ontology[8] where the DCES core properties (i.e., without qualifications) are implemented as datatype properties. In this way the model semantics are better captured; however, properties' fillers are deemed to be plain literals.

No matter how the DC ontology may be implemented, the burden of producing from scratch a whole new set of richer descriptions can be prohibiting, since it usually requires a huge amount of manual effort [16].

Therefore, in this section, we propose an implementation of a DC ontology that is to be carried out in terms of a most centralized approach. To do this we are based on the semantic profiling technique, well applied previously on fully structured knowledge domains, such as the CIDOC-CRM [1]. Using this technique we try to better capture the intended semantics of the DC metadata domain, having the DC RDF(S) schema as a starting point. Our goal is to upgrade this ontology up to OWL and OWL 2 level [25], by incorporating new constructs and refinements, available only in these languages. At the same time, we build upon the initial model and do not require any alternations in its original specification. In this process, we also take into account the LOM metadata, with which we have extended the original DSpace schema. The resulting ontology, including the new refinements, is then populated in an automated way from metadata already existing within the live DSpace installation of the University of Patras institutional repository, through its OAI-PMH interface (see Sect. 3.5).

Following this approach we therefore achieve to:

- Respect and reuse the original DC specification by providing a semantic application profile suitable for the particular domain's needs.
- Keep the repository's internal data model intact by employing its interoperability facilities (OAI-PMH).
- Reuse existing descriptions in an automated way, transparent to the end user, without any demands for additional annotations.

Implementation steps can be organized in three phases, namely, *Syntax Transformation*, *Semantic Intension* and *Semantic Refinement*. Figure 2 shows the interaction and flow between the various steps that follow.

3.2 Syntax transformation

An important variation from the implementation presented in [15] is that here, we do not have a well-structured knowledge domain as the one expressed by CIDOC-CRM, but our instances are virtually a flat aggregation of elements. The DC

[6] http://www.aifb.uni-karlsruhe.de/WBS/rvo/ontologies/.

[7] http://protege.stanford.edu/plugins/owl/dc/protege-dc.owl.

[8] http://orlando.drc.com/semanticweb/Topics/Ontology/Ontologies.asp.

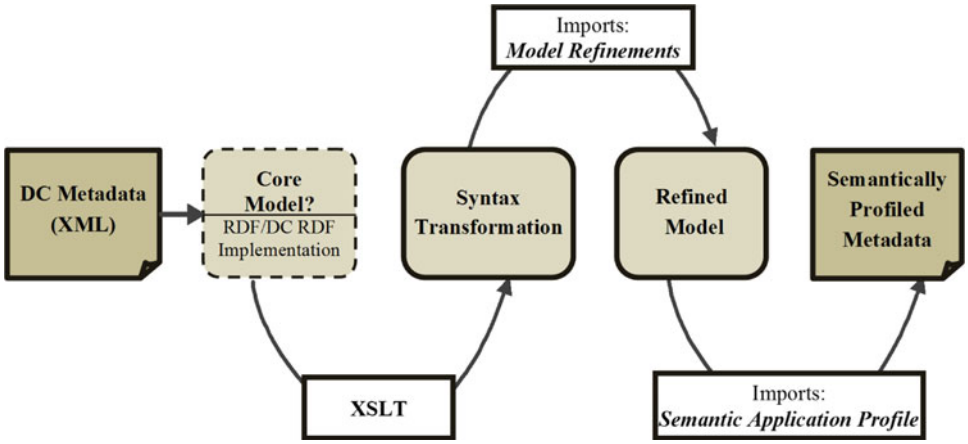


Fig. 2 Implementation steps of the DC ontology

RDF implementation may be a good starting point, but there is still a gap to be filled in order to reach this state. Therefore, we need a transition from an absent semantic model to an item-property-predicate situation, thus invoking the RDF semantics. This transition is offered by an XSLT, which can be considered part of the syntax transformation phase in the semantic profiling process.

This transformation amounts to:

(a) Minimal syntax transformation to "clean-out" OAI overhead statements. For example XML elements `<responseDate>` and `<request>` are OAI-specific and are not required. The same holds for records headers.

(b) Assign types (datatypes) to literal values. DC offers an abstraction for datatypes, called *syntax encoding schemes*. In [3] the use of these schemes is *optional*, i.e., it is a recommended best practice and the ranges of corresponding properties amount just to `rdfs:Literal`. However, these schemes are in fact equivalent to the actual XML Schema datatypes that are more likely to be supported in OWL applications, since in OWL 2 it is specified that an application may signal an error if an unsupported datatype is encountered [22]. For example, `dcterms:W3CDTF` corresponds to `xsd:date`, `dcterms:RFC3066` to `xsd:language` and so on. These datatypes have to be explicitly assigned to literal values, since OWL needs explicit datatyping of resources, even when the corresponding property's range has been defined as an `xsd`: datatype.

(c) Reify literals (as objects) originating from specific (controlled) vocabularies, such as MIME types. In particular, DSpace relates items with literals corresponding to MIME types through the `dcterms:format` property. The use of MIME types as fillers to `dcterms:format` is also a DCMI recommended practice [3]. By this approach we are able, for example, to query the ontology about items of a certain MIME type. These MIME types are in fact instances of classes that model categories of MIME types in a hierarchy [see also step 3.4(a)].

For example, in the case of an MS Word document we specify

```
<dcterms:format rdf:resource=
    "&dspace_ont;msword"/>.
```

In addition to MIME types, we also reify literals representing LOM-specific values, such as typical learning times (e.g., `lom:one_day`, `lom:one_semester`) and intended end user roles (like `lom:author` and `lom:student`). Likewise, literals corresponding to DSpace-specific content types are reified; for example, in the case of a Powerpoint presentation we specify

```
<dcterms:type rdf:resource=
    "&dspace_ont;Presentation"/>.
```

Further, each reified object is annotated, through the `rdfs:label` property, with literals representing its multilingual translations. To every such annotation an `xml:lang` attribute is assigned, taking advantage of the fact that translations are stored in pairs of n, where n is the number of languages supported.

(d) Reify other literals to be able to identify and express potential semantic relations. For example, DSpace author names are defined as objects and their comma separated first and last names (as stored by DSpace) are parsed and assigned to corresponding properties from the popular FOAF ontology.[9] We preserve the language of relevant text values, using the `xml:lang` attribute. However we cannot relate translations, for example, between author names or item titles, since this relation is lost as soon as the user enters this information in

[9] http://xmlns.com/foaf/spec/.

DSpace (the underlying DB schema does not provide for this).

(e) Introduce and re-assign namespaces. This means that the traditional DC elements namespace[10] (corresponding to the `dc:` prefix) is replaced by the DC Terms namespace[11] (`dcterms:`). This substitution, even for traditional DC elements, is unavoidable in order to achieve compatibility with the DC RDF implementation [3]. We also introduce a new namespace for the DSpace-specific elements,[12] which is associated with the prefix `dspace-ont:`.
Finally, we map LOM elements to DC Terms properties. This mapping is based on the LOM-to-simple-DC mapping suggested in [12], on the work described in [13], which proposes a potential LOM to DCAM mapping, and on the semantic interpretation of both the LOM and qualified DC elements. LOM concepts are prefixed with `lom:`, which corresponds to their official namespace.[13] For example, we map `lom:intendedenduserrole` to `dcterms:audience` and `lom:annotation` to `dcterms:description`.
This step may also be considered as a phase on its own (phase 0: *compatibility phase*), which is unavoidable in order to achieve compatibility with the DC RDF implementation.

Most of these steps are implemented in an XSLT document ('transformer.xslt'[14]), which is used for the population of the ontology, as described in Sect. 3.5.

3.3 Semantic intension

Second, we can commence with the semantic intension of the DC model. To do this we need to establish (or devise) a core model first. Such a model is

- offered by the RDF semantics, inherent in the DC RDF implementation,
- further profiled by the new DC implementation that incorporates DCAM as well as domain and range restrictions,
- further profiled by post-RDF characterizations of properties, like inversity, symmetry, transitivity, functionality. For example, we define `dcterms:relation` as symmetric and `dcterms:hasPart` as the inverse of `dcterms:isPartOf`.

The result of this phase is the creation of an OWL document that imports the original DC Terms RDF ontology ('dcterms.rdf'[15]) and contains the above refinements ('dc-ont.owl'[16]). This document comprises the DC ontology, expressed in OWL format, which refines the original specification by utilizing OWL-specific constructs, but retaining at the same time its interoperability. The class hierarchy of this ontology is depicted in Fig. 3.

3.4 Semantic refinement

Now, it is time to add semantic refinements for our particular application scenario, i.e., the University of Patras institutional repository. This is conducted by:

(a) Model vocabularies in subsumption hierarchies. For example, the MIME type vocabulary that DSpace supports is implemented as a partition of subclasses, with instances, and is related to `dcterms:FileFormat`. For this implementation, we took into consideration both the existing online note (RFC) about MIME types[17] as well as the DSpace MIME types registry. As an example, we define the class `dspace-ont:Application` where the instance `dspace-ont:msword` belongs to.

(b) Identify and represent DSpace notions of "item", "collection", and "community" as classes. We refer to individuals belonging to these classes with their (unique) handles, as assigned by DSpace. For example:

```
<dspace-ont : Collection rdf : about =
  ″&dspace-ont; hdl_1987_42″/ >
```

In addition, collection is defined as a subclass of community, capturing the notion that collections refine communities in DSpace. Items are related to collections with the `dcterms:hasPart` property (remember that `dcterms:hasPart` is defined as the inverse of `dcterms:isPartOf`):

```
<dspace-ont : Item rdf : about =
    ″&dspace-ont; hdl_1987_57″ >
  <dcterms : isPartOf rdf : resource =
  ″&dspace-ont; hdl_1987_42″/ >
</dpace−ont : Item >
```

[10] http://purl.org/dc/elements/1.1/.

[11] http://purl.org/dc/terms/.

[12] http://ippocrates.hpclab.ceid.upatras.gr:8998/dc-ont/dspace-ont.owl#.

[13] http://ltsc.ieee.org/xsd/LOM.

[14] http://repository.upatras.gr/dspace/dc-ont/transformer.xslt.

[15] http://repository.upatras.gr/dspace/dc-ont/dcterms.rdf.

[16] http://repository.upatras.gr/dspace/dc-ont/dc-ont.owl.

[17] http://www.ietf.org/rfc/rfc1341.txt.

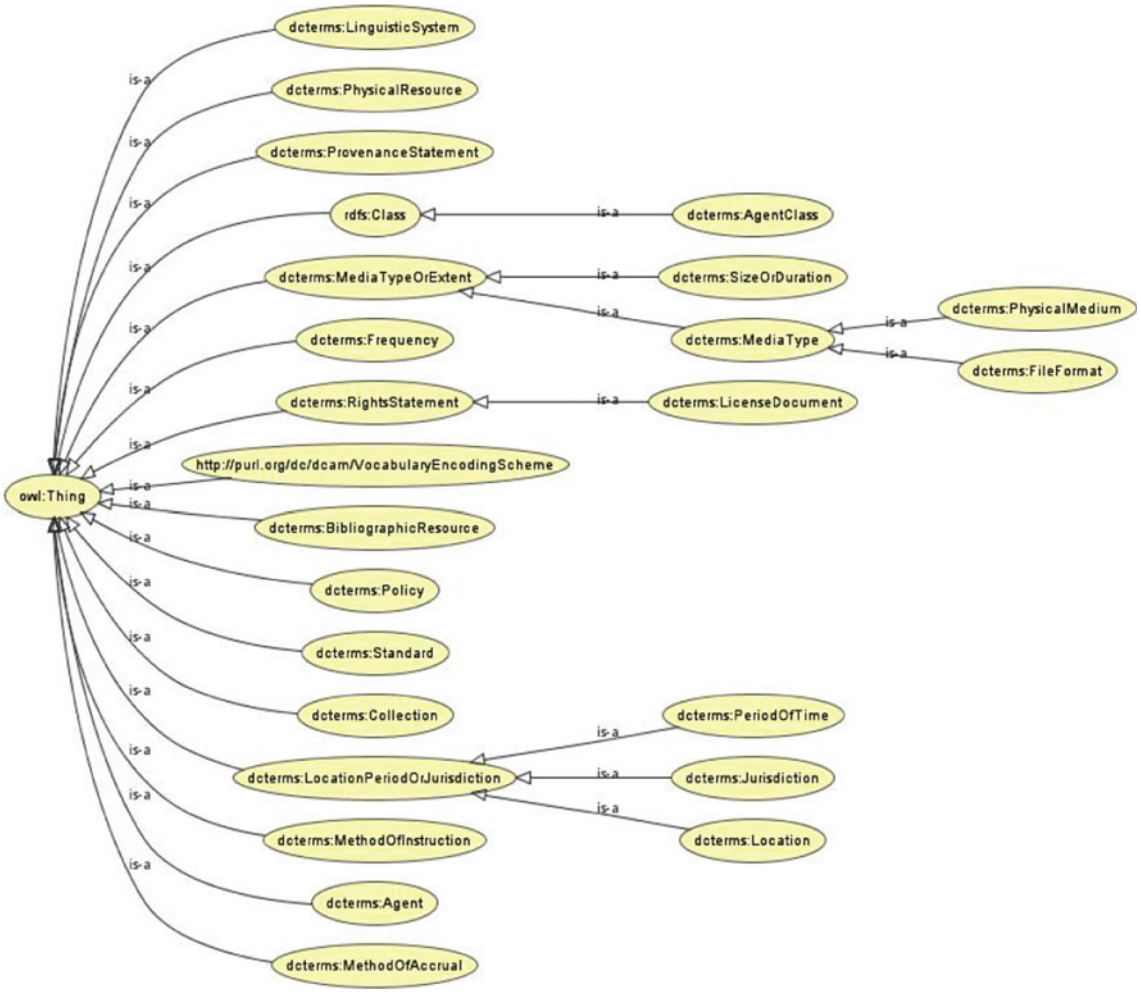


Fig. 3 Class hierarchy of the DC ontology, as shown by the OWLViz Tab of Protégé 4.0

(c) Identify and represent the non-DC notions of "author" and "sponsorship" and relate them to the initial model: we define, under a different namespace, author and sponsorship as OWL object properties, and make them sub-properties of `dcterms:contributor` and `dcterms:description` respectively.

(d) Model complex relations using role-forming operators and restrictions ("some" and "for all") available in OWL 2: For example, we define the notion of "co-author" in Manchester Syntax (see Sect. 4), as

```
inv(author) o author SubPropertyOf
    co_author
```

Then, the co-author property should hold between every two authors of the same item. This declaration implies that a particular author will be always related with itself through the co-author property. This, however, is an unavoidable side-effect coming from the fact that DSpace makes no restriction between first and subsequent authors.
Moreover, we refine sponsorship by

```
inv(dcterms:contributor) o sponsorship
        SubPropertyOf sponsorship,
```

meaning that also the authors of items are receiving sponsorship by the same institution.
We also state that

```
not(Item) and (dcterms:hasPart some
    Item) SubClassOf Collection
```

meaning that everything that has part an item is a collection, except if it is an item itself. Also collections can only have items as parts:

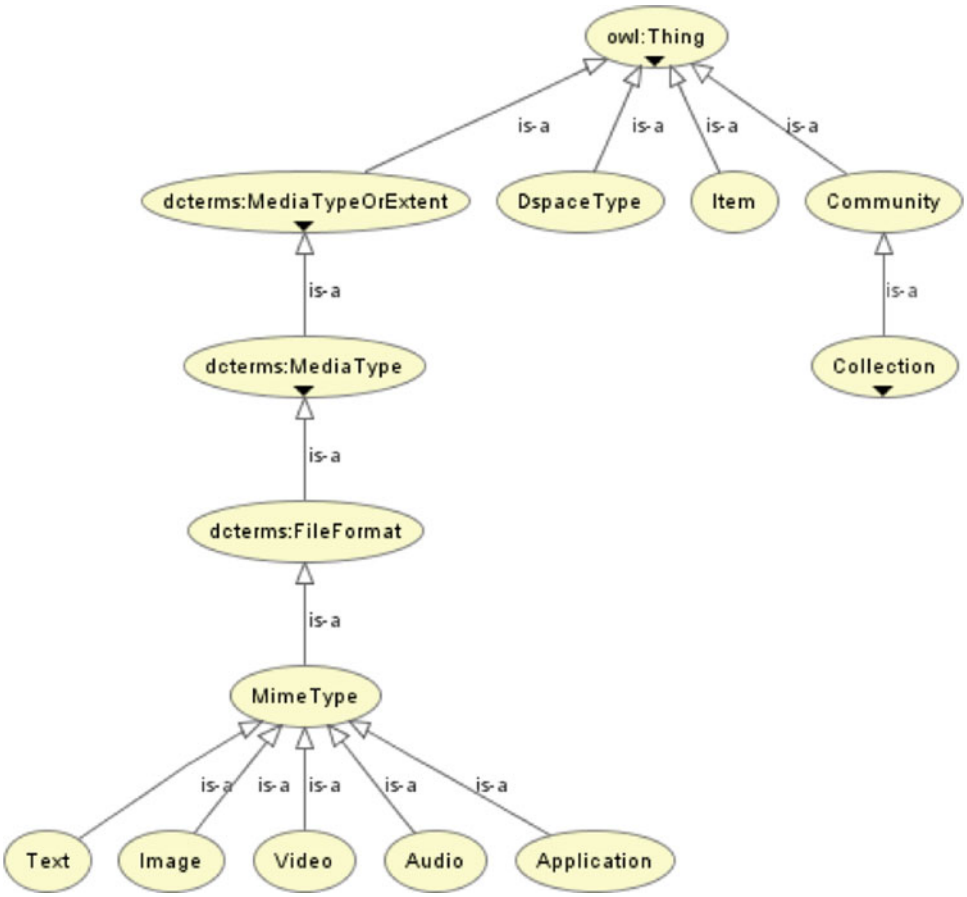


Fig. 4 Partial view of the DSpace ontology class hierarchy (excluding some imported axioms)

`Collection` **`SubClassOf`**
(`dcterms:hasPart` **`only`** `Item`).

Finally, we can model some additional, common-sense facts, which may involve notions in more than one ontologies: for example, a LOM learning object of type 'Slide' would reasonably be of type 'Presentation' as well. This can be represented in the following subclass relation:

(`dcterms:type value lom:Slide`)
`SubClassOf` (`dcterms:type value dspace-ont:Presentation`)

The above declaration simply means that everything that has `dcterms:type lom:Slide` would also have `dcterms:type dspace-ont:Presentation`.

This phase results in creating a new OWL 2 document ('dspace-ont.owl'[18]) containing OWL constructs that refine the DSpace data model in a semantic manner (see Fig. 4) and importing all other ontologies.

At this point, the semantic profiling technique foresees the need to accommodate other additional schemata that may be found suitable for a more precise representation of our particular application's problem domain. As mentioned, in the University of Patras DSpace installation, we have also enriched the original DC schema with LOM metadata in a way to better convey the educational characteristics of resources. Therefore, in this phase we also create an ontology model representing (part of) the LOM schema: we create OWL classes grouping LOM vocabulary values and, when it is semantically consistent, we relate them to DC Terms classes (domains and ranges). For example we declare `lom:TypicalLearningTime` to be a subclass of `dcterms:SizeOrDuration`. The class hierarchy of this separate LOM ontology is depicted in Fig. 5.

The LOM-specific constructs are kept in a separate document ('lom.owl'[19]), which is imported by dspace-ont.owl.

3.5 Ontology population

In the previous sections we have described the process for creating the ontological model that we use in our application,

[18] http://repository.upatras.gr/dspace/dc-ont/dspace-ont.owl.

[19] http://repository.upatras.gr/dspace/dc-ont/lom.owl.

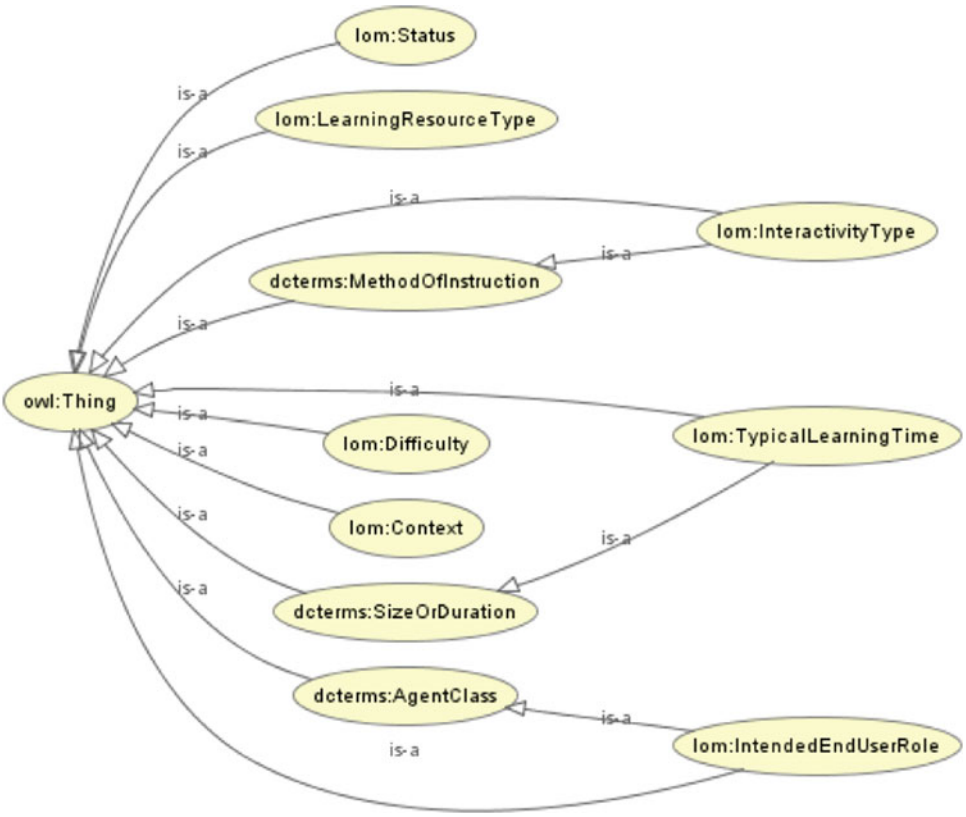


Fig. 5 LOM ontology class hierarchy

by creating a semantic application profile of the qualified DC ontology, tailored for our repository's domain. The next step is to populate this ontology with descriptions coming from the University of Patras institutional repository.

DSpace offers the ability to harvest its metadata by providing an OAI-PMH interface. Internally it follows a particular application profile, borrowing heavily from the Library Application Profile [2]. Including qualifications, DSpace fosters a total of 66 elements, not all of which are visible to the user.

OAI-PMH is an HTTP-based protocol for interoperable metadata harvesting [19]. It offers a means to collect and expose the repository's content in a standardized way, thus rendering the latter's metadata available to service providers. OAI-PMH gathers the repository's database objects, replicates them and finally produces an XML file out of the "harvested" object's metadata. Though it would be easier to access the repository's database directly, we opted for OAI, precisely in order to show in practice how our approach can be generalized to maintain the interoperability of our implementation.

The repository's OAI-PMH facility is configurable as to what elements are to be exported (including their namespace and label) and also supports simple DC (oai_dc) as well as its qualifications (qdc). Part of the syntax transformation phase is implemented exactly in this configuration: First, we configure OAI to export metadata also in qualified DC format, as richer from a semantic point of view. Then, in the appropriate configuration file, we re-assign namespaces and map LOM and simple DC elements to corresponding DC Terms properties (step (e) of the syntax transformation phase, see Sect. 3.2). As a result, we are able to harvest as much of the qualified DC elements as possible and achieve at least a *syntactic* level of compatibility with our ontology model.

A series of syntax transformations, which we have implemented in an XSLT file and described in Sect. 3.2, tries to better capture the semantic relations implied in these metadata as well as to construct the OWL-specific instantiations, thus achieving a *semantic* level of compatibility with our ontology. Indeed, a servlet is responsible for the dynamic construction of the populated ontology, by automatically harvesting the repository's metadata to an XML file through OAI and transforming it according to the XSLT document. The resulting OWL ontology imports the semantic application profile ('dspace-ont.owl') as well as the LOM-specific constructs ('lom.owl'). The servlet's response is then fed to the semantic search and navigation services that we describe in the following section.

4 Semantic search and navigation

In this section we discuss the design decisions and the implementation of the semantic enhancements to the DSpace

digital repository system. These decisions dictate, to some extent, the implementation carried out and the specific functionality provided. A brief introduction to this functionality has also been given in [17].

First we give an overview of these principles and document their necessity and reasoning behind them. Then we present an outline of the implemented architecture and put it in to perspective by displaying how it wraps around the DSpace system. Finally we describe the new functionality being made available, which mostly revolves around two main services: Semantic-enabled search of DSpace content and semantic navigation among the repository's instances.

Our intention is not to make an exhaustive account of the implementation and its technical details; rather this section gives an insight of the design process and helps to evaluate the functionality provided as well as to better understand the potential of Semantic Web enabled digital repositories.

4.1 Design goals

Most of the design decisions stem from a set of requirements that where posed beforehand, in order to guarantee reproducibility and applicability of our efforts, as well as to ensure the potential of the semantic services offered. Regardless of the specific way in which they are implemented, these goals and requirements should be unanimously taken into account before developing a semantic digital repository and can be summarized as follows:

4.1.1 Interoperability

Given the various ways data sources are often structured and communicated by different systems, a high level of interoperability is of crucial importance. At its core, interoperability is achieved by adhering to information standards, at any level: Repository's metadata are structured according to the XML format, ontology models are represented using W3C's OWL and OWL 2 specifications, and the semantic gap between the two is bridged using an XSLT transformation. Further, this transformation upgrades interoperability to a semantic level, by rendering the repository's metadata descriptions semantically compatible to the ontology's structures. Lacking a communication protocol for the exchange of OWL information, we at least opt for wrapping and transforming OAI-PMH responses that offer a standard way for harvesting metadata from data providers. This comes in contrast to accessing the repository's database directly, as this could be dependent on the proprietary DB schema. As a result, our approach could be seamlessly integrated with other resource management systems, at least OAI compliant ones.

4.1.2 Support for OWL 2

The need to support this newly proposed extension to OWL comes from the fact that OWL 2 is able to represent a richer set of semantics than its predecessor, thus enabling more advanced inferences. For example, in Sect. 3.4 we make extensive use of the ability to combine property expressions in chain-forming axioms, a particular characteristic of OWL 2 only. On the other hand, this reduces our choices of inference engines to only supporting ones, such as FaCT++ and Pellet. Performance is not our main concern here, since the OWL 2 reasoning algorithm is known to be scalable [11] and its implementation in these reasoners is heavily optimized [29]. However, we are forced to use a direct in-memory implementation, since none of these reasoners supports a communication interface capable of supporting OWL 2 expressivity. DIG[20]—a protocol for communicating with DL reasoners—as it is currently supported by FaCT++ and Pellet, is incompatible with OWL DL, let alone OWL 2. Only DIG's latest specification, called OWLlink [20], has been aligned with OWL 2, but this version is not yet supported by the aforementioned reasoners. Therefore, high expressivity comes at the cost of a truly distributed 3-tier architecture.

4.1.3 Extensibility

A major design decision was to totally implement our extensions using the OWL API, while avoiding references to DSpace-specific methods and classes. OWL API equips us with a satisfactory layer of abstraction on top of which further extensions can be implemented. In addition, it does not restrict us to a particular inference engine or a specific reasoning approach: The selection of the reasoner class used can be easily parameterized, while the reasoning strategy can stay the same, rendering our implementation reasoner-independent. Furthermore, it is easy to support other querying protocols and/or methods: In our implementation, a query is given in the form of a Manchester Syntax class expression [10]. Just as easily, query formulation can be extended to follow another paradigm, such as SPARQL/OWL [28], as soon as its specification grows mature and a supporting parser is implemented.[21]

4.1.4 User-friendliness/intuitiveness

Unfortunately, a common way for querying OWL knowledge bases has not been standardized yet. SPARQL is a language for querying RDF graphs, but it lets out of scope the inference capabilities, taxonomic queries, and the particulars of the essential ontology languages OWL and OWL 2. A subset

[20] http://dig.cs.manchester.ac.uk/index.html.

[21] http://wiki.webont.org/page/SPARQL/OWL.

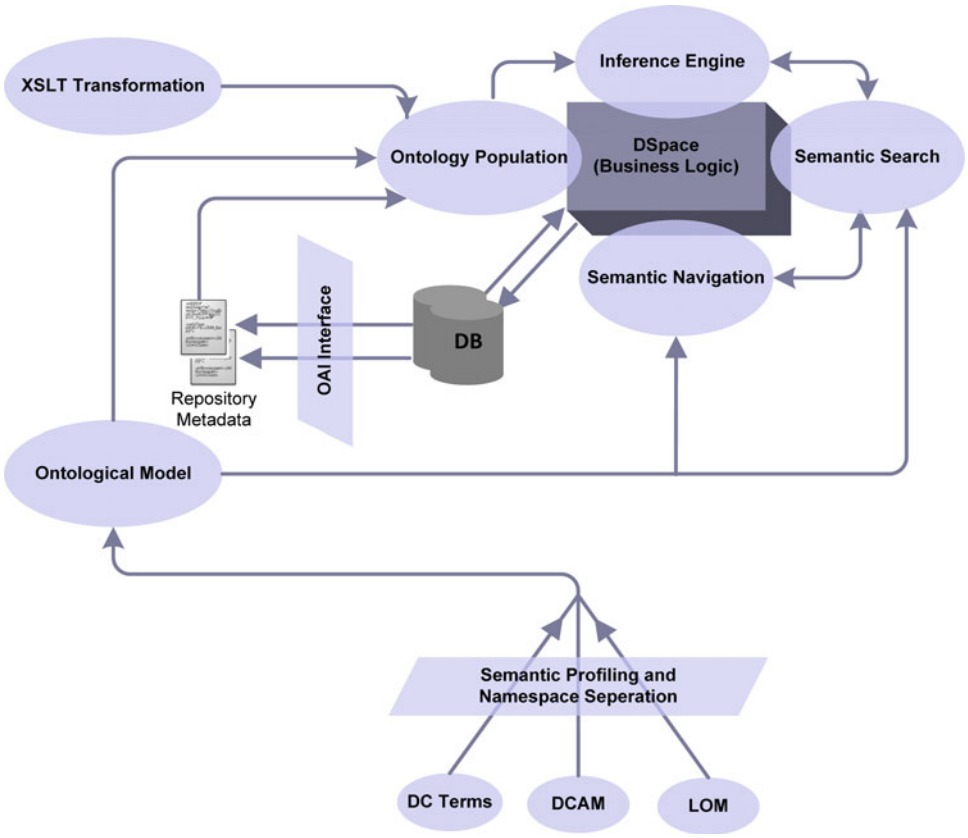


Fig. 6 Architectural model of the semantic search extension of DSpace

of SPARQL that allows, under certain conditions, the expression of conjunctive queries and their combination with taxonomic ones is SPARQL-DL [28]. SPARQL-DL is associated with the ontology language OWL DL and it can be evenly extended toward OWL 2, by adding corresponding query atoms like `Reflexive(p)`, for checking the reflexivity of a property `p`.

Therefore, in order to build a "user-friendly" service and keep the complexity of query formulation as low as possible, we implement a query interface based on class construction using Manchester Syntax, in a way inspired by the DL Query Tab of Protégé. Query results are formed by all individuals that are inferred to be instances of the constructed class. Manchester Syntax has the advantage to offer a pseudo-natural English language expression of classes, thus facilitating, to some extent, the end user to formulate a query.

Besides, the process of querying knowledge bases—and consequently the implemented service—is mostly addressed toward those users that are more familiar with Semantic Web technologies. So the use of a less complex syntax, as compares to other OWL syntaxes, that employs simple words instead of complicated mathematical symbols can definitely offer significant help toward this direction. In addition, we have tried to make the process of query formulation intuitive, by implementing an AJAX-based suggestion and auto-complete mechanism, where matching entities names are suggested to the user, as the query is typed in.

4.2 Architectural overview

An overview of the services we have built around DSpace is depicted in Fig. 6. The most important modules and interfaces that enable semantic services in our digital repository are the following:

- *Semantic Search* interface, which, in collaboration with the appropriate inference engine, allows for the construction, submission, and evaluation of a semantic query. Retrieved results are displayed here in the form of a list.
- *Semantic Navigation* interface is where detailed ontological information about a selected entity (individual) are presented.
- *Ontology Population* refers to the dynamic construction of the ontology, which comes from DSpace's OAI harvested metadata, after applying the appropriate XSLT transformation on them, as described in Sect. 3.5.
- The *Inference Engine* is responsible for processing the ontological documents and for performing reasoning over them.

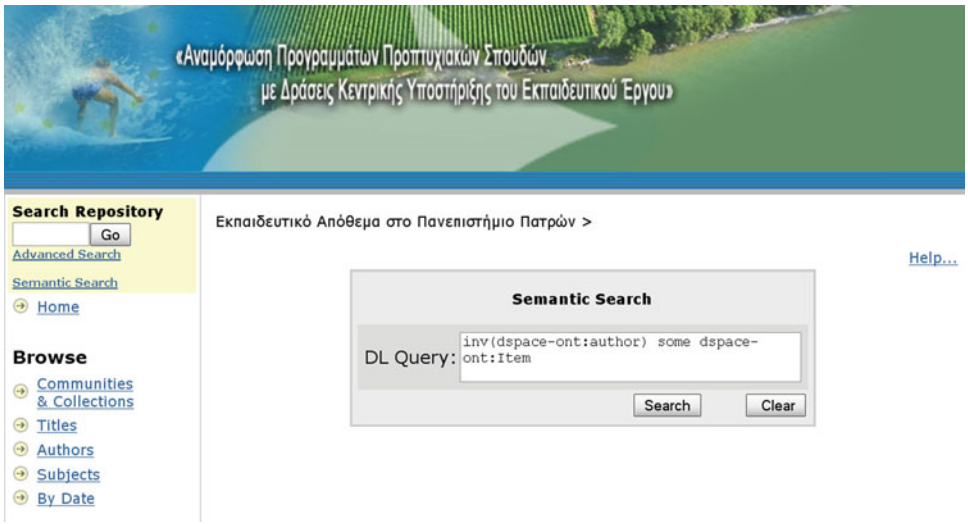


Fig. 7 Semantic search interface

These facilities have been implemented in Java servlets, using the OWL API. These servlets extend `DspaceServlet`, an extension of the Java `HttpServlet` class. This is the only (and necessary) reference to the DSpace API. Reasoning is performed at query time by the FaCT++ inference engine, but any other DL reasoner may be used, as stated in Sect. 4.1. In any case, the underlying reasoner loads and classifies the populated ontology whenever the corresponding servlet is called. FaCT++ is interfaced by the appropriate abstract class of the OWL API `(Reasoner)`, through JNI.

The populated ontology is dynamically constructed and silently fed to the reasoner over HTTP. In fact, our semantic search and navigation services are designed and implemented in such a way that they can work with any OWL ontology, not just the one populated with the repository's metadata: Since the ontology URI is passed as an HTTP parameter, it is easy to parameterize the user interface to ask for an ontology URL as well, making our implementation totally independent of the specific ontological model.

This 'plug-in' architecture is further enhanced by the fact that, in our serlvets, we avoid any references to DSpace-specific code or any direct access to DSpace's database. In this way we ensure that our services are also system-independent. However, context with DSpace itself is indirectly maintained, since it is still possible to open DSpace's simple item view page from within the navigation pane (see the following section).

4.3 Functionality

In the following, we describe our extensions to the DSpace system in terms of their main functionality and implementation characteristics.

4.3.1 The semantic search interface

A web-based, graphical user interface has been implemented that supports semantic query construction and validation, result retrieval, and presentation.

Queries are typed as simple text in the provided textarea field (see Fig. 7). Query formulation is based on the Manchester OWL Syntax. Therefore, accepted queries should be valid ontological class names, class expressions (e.g., existential quantifications, cardinality restrictions, and so on), or Boolean combinations of class expressions. The entered expression is then parsed and evaluated to an OWL Description, namely, a (possibly anonymous) class, using the Manchester OWL Syntax Editor Parser, available from CO-ODE. Results retrieved by semantic search are exactly the instances of this class, including both asserted as well as inferred instances.

Class, property, and individual names participating in a query should be relevant to the specific ontology. For example, DSpace main entities item, collection and community map to classes that are specified under the DSpace-specific elements namespace we have defined (prefixed by `dspace-ont:`). Dublin Core fields map to properties that use the prefixes `dc:` and `dcterms:`. It is important to note, though, that properties with different prefixes are different properties. For example, `dc:format` is different than `dcterms:format`. For this reason the query interface is adjusted to accept *QNames* for entities participating in a query.

Since users do not know in advance the contents of the ontology, an auto-complete facility is provided by the semantic search interface in a manner similar to the CO-ODE Ontology Browser.[22] While a word is being typed, a list of entities contained in the knowledge base (class, property, and indi-

[22] http://owl.cs.manchester.ac.uk/browser/.

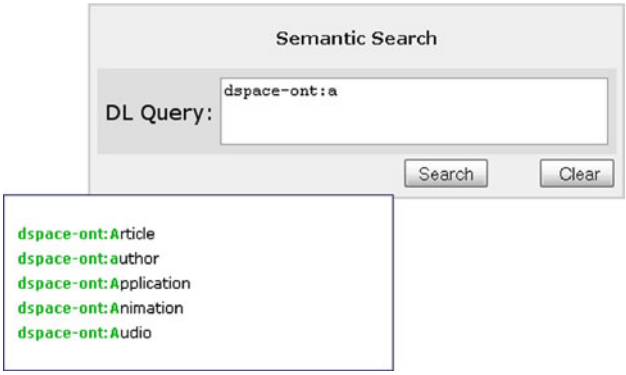


Fig. 8 Auto-complete facility

vidual names) is suggested, so as to match the current part of the entered word. This list has been created in advance, right after the ontology loading. In fact, the QNames of these entities are used, utilizing a prefix-to-namespace map constructed out of every imported ontology. The user may continue typing or take advantage of the auto-complete facility and select one of the suggested entities.

For example, as shown in Fig. 8, there are three different suggestions when typing `dspace-ont:a` and clicking on one of them leads to the automatic completion of the query with the selected entity name.

After query evaluation, search results are displayed in the form of a list. Retrieved entities are organized in pages, each containing at most twenty results. Users navigate through the results by following a "Next Page" or "Previous Page" link.

It is important to note that results returned by semantic search are not merely DSpace items, as is the case with the traditional search facilities of DSpace. They can be individuals of any type. For example, we can ask for the names of all authors as individuals, something which is stated in a query in the following way:

```
inv(dspace-ont:author) some dspace-ont:
    Item
```

In addition, search results appear as links that, when clicked, redirect to a new page containing a detailed view of the selected entity's ontological information.

4.3.2 Individual view and navigation pane

As mentioned above, query results are linkable entities and when clicked a new page is created for them on the fly. This page gives a detailed view of the selected individual and allows navigation among other instances which this individual may be related to.

More specifically, the main role of this interface is to give a detailed reference about the individual's ontological information, organized in a way that is depicted in the main window of Fig. 9. Its main objective is to give information about the following:

- *Classes* to which the selected individual belongs.
- *Object properties* that relate the current individual to other individuals.
- *Data properties* and *annotation properties* that reveal the individual's relations with text (literal) values, which do not correspond to individuals.

The name of each displayed class is a link that, if selected, the user is presented with all the instances of this class. In fact, the user is redirected back to the semantic search page with the specific class already predefined.

Object property values (which are individuals themselves) can also be clicked. This leads to the dynamic construction of the individual view and navigation pane for the particular entity. In addition, if the current individual is found to be reversely related to other individuals, then these relations are displayed as well. This is done by prefixing the displayed object property with the word "inverse" or by just displaying its named inverse, if it happens to have one. Moreover, negative object and data property assertions are also taken into consideration and may appear in the result table, if existent.

For the data property and annotation values, their datatype or their language identifier is shown in the information table. If a value has a datatype of `xsd:anyURI`, then it is considered to be a link. It is this feature that helps to maintain context with the original DSpace item view, through the `dcterms:identifier` property, which may link an individual to its handle.

Figure 9 gives an intuitive view of how users can navigate among ontological details, as described above.

5 Evaluation and examples

Semantic search is mainly addressed toward those repository users that have some basic knowledge about Semantic Web technologies. In this sense, we argue that the semantic search and navigation services we have plugged into the University of Patras repository[23] add value to the traditional search mechanism, at least in four ways:

1. They allow retrieval and presentation of entities of any type, not just repository items.
2. They can augment traditional search, by retrieving more results, for a given query.
3. They improve searching precision, by fetching more accurate results.

[23] http://repository.upatras.gr/dspace/semantic-search.

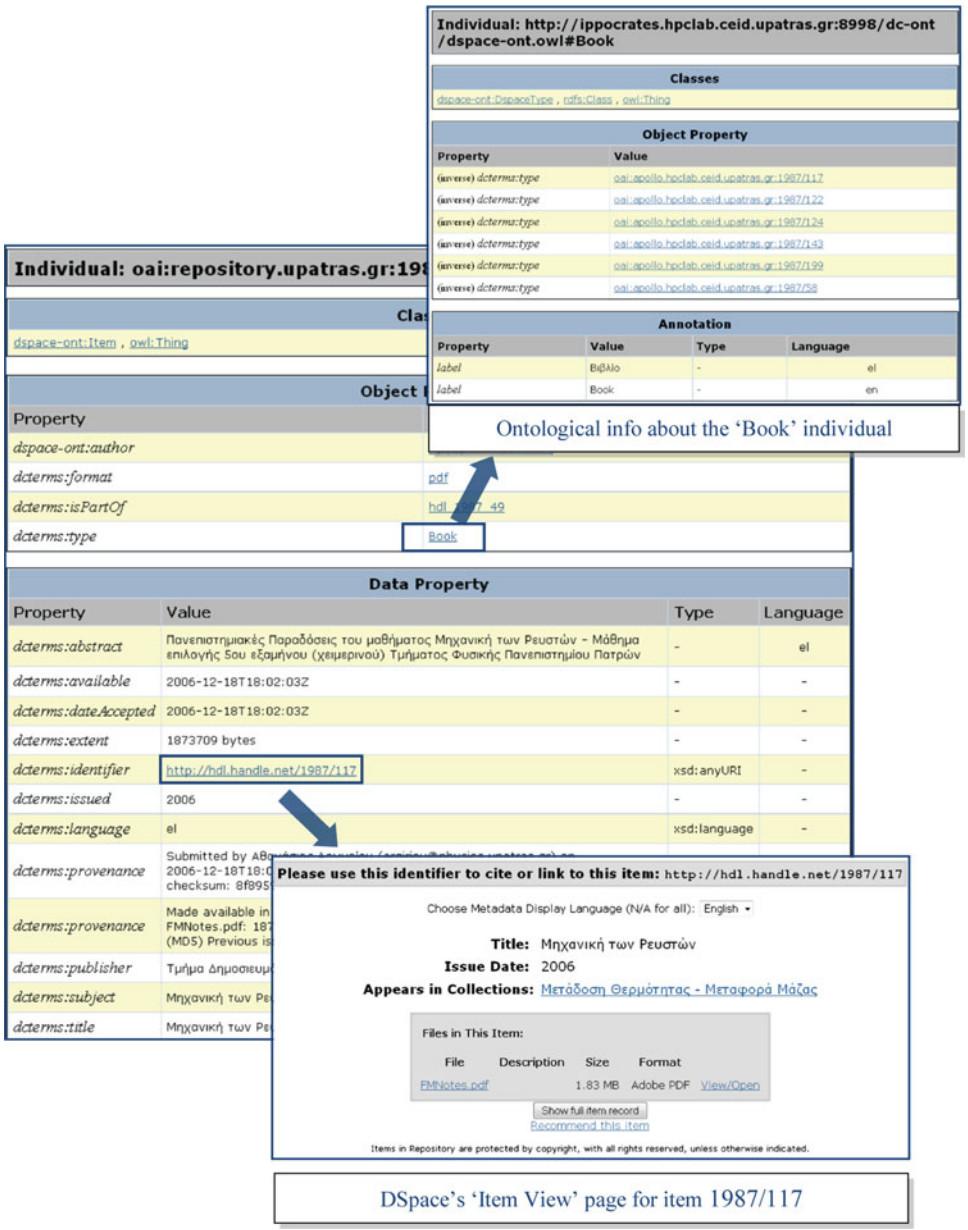


Fig. 9 Individual view and navigation pane for item 1987/117

4. They allow the expression of queries that is impossible to be conducted just by keyword-based search.

The evaluation of semantic search services is still a problematic process and known IR evaluation patterns (like *recall* and *precision*) do not apply well in this field (for example see [7]). Therefore, in this section we first make a quantitative analysis of our ontology and point out the role that its particular expressivity characteristics have in semantic search. Afterward, we present some semantic query examples, indicative of the aforementioned points, putting emphasis and evaluating their results from a qualitative point of view. Finally we discuss how our implementation would perform when handling large-scale ontologies.

5.1 Ontology metrics

In order to create our ontological model we had to declare a number of new classes and properties, as well as to formulate a number of necessary axioms. The exact amount of OWL constructs, which we made use of, is summarized in Table 2. In this table we give the number of classes, properties, and individuals that compose each OWL document we have created. We also include the exact amount of declared axioms.

Table 2 Number of OWL constructs and axioms used in: *dc-ont.owl, dspace-ont.owl* and *lom.owl*

	Counts	dc-ont.owl	dspace-ont.owl	lom.owl	Total
Metrics	Classes	–	14	11	25
	Object properties	13	8	–	21
	Data properties	–	–	–	–
	Individuals	–	58	45	103
Class axioms	Subclass	–	11	10	21
	Equivalent	–	2	–	2
	Disjoint	–	3	1	4
	General Class Inclusion	–	2	–	2
Object property axioms	Sub-property	–	3	–	3
	Inverse	6	–	–	6
	Transitive	2	–	–	2
	Symmetric	1	–	–	1
	Chain sub-property	–	3	–	3
Individual axioms	Class assertion	–	57	45	102
	Different individuals	–	1	–	1

Table 3 Statistical information about the populated ontology

	Counts	Populated ontology
Metrics	Classes	47
	Object properties	73
	Data properties	35
	Individuals	399
Individual axioms	Class assertion	283
	Object property assertion	320
	Data property assertion	897
	Different individuals	1

Table 4 OWL and OWL 2 constructs used

Notion	Usage
Classes	Represent DSpace structural elements
Object properties	Represent DC and non-DC relations
Characteristics of properties	Discover relations between individuals (transitivity, symmetry, etc.)
Individual axioms	Define characteristics of individuals
Subsumption hierarchies	Organize classes in hierarchies
OWL 2—specific constructs	
Punning	States that a name can be treated either as an individual or a class
Role chains	Combine property expressions in chain-forming axioms

In Table 3, we summarize similar information (number of classes, properties, and individuals) about the dynamically populated ontology that our services finally act upon. Note that this ontology formally imports all the ontologies mentioned in Table 2 as well the DCMI RDF ontology.

All these metrics offer a relative measure about the size of the ontologies used. However, they do not say much about their complexity, as some OWL axioms are traditionally "harder than others" [5]. For example role chains, transitives, and inverses, despite their low count, are nevertheless hard to compute when they exist in certain combinations. At the same time though, they offer a considerable "expressivity boost" by allowing the inference of hidden and implied relations and by enabling some unique queries, as shown in Sect. 5.2. Therefore, it is not in the amount of axioms but in their qualities, the extent to which a certain ontology (including our own) can adequately capture user needs.

We also notice that the individual count is not very high. This is due to the fact that the items existing in the University of Patras institutional repository are currently only a few; however, this has no effect on the quality of the conducted inferences, as the soundness of the results does not depend on the amount of items. This is because the queries are based on sound and complete reasoning algorithms implemented in the underlying inference engine and the results produced are strictly deterministic. Scalability issues are discussed in Sect. 5.3.

In Table 4 we group all OWL notions used in the formulation of the final ontology document. Constructs particular to OWL 2 are also pointed out.

5.2 Qualitative analysis

In this section we show how the implemented semantic search and navigation facility, fed with the semantically enriched

DC ontology, can offer additional and improved search and knowledge discovery services on top of a live digital repository system. We actually present some example queries that suggest the basic gains of the semantic search facility, and then we comment on the obtained results.

In comparison to the traditional repository search, the queries we present can be divided in four categories:

- Queries that are semantically equivalent to the traditional search queries.
- Queries that allow retrieval of entities other than items.
- Queries that produce improved results over traditional ones.
- Queries that infer new and implicit knowledge that is impossible to be retrieved otherwise.

Next, we present simple examples in each category. We argue that these kinds of queries can match the needs of users that expect the added value reasoning-based services have to offer; at the downside comes the (relatively low) cost of familiarizing with a syntax as verbose as natural language may be, but which can also get arbitrarily complex.

5.2.1 Equivalent queries

Semantic search is equivalent to the existing DSpace search mechanisms, at least in two aspects: the kind of queries that can be posed and the result retrieval capability. Although it applies to a more expert group of users, it offers the means for expressing all kind of queries that can also be conducted by keyword-based and advanced search.

As an example, suppose we want to retrieve items corresponding to authors that have a particular surname (e.g., *Dawson*). Using the advanced DSpace search service, we just supply the keyword 'Dawson' and obtain one result. With semantic search, we have to formulate the query like this:

```
dspace-ont:author some (foaf:surname
  value "Dawson")
```

In this case we also retrieve one result, the same as previous.

Suppose, now, we want to seek for all those items that are of type 'Presentation' and have 'Students' as their target audience. With advanced search, we pose our request by giving the appropriate keywords in the corresponding fields, thus retrieving three results. With semantic search, we have to write the following query:

```
dcterms:type value dspace-ont:
  Presentation and dcterms:audience
  value lom:Student
```

Returned results are also three in total and coincide with those we got by the advanced search service.

5.2.2 Entity retrieval

In DSpace, the main information unit is the *item*, which represents a specific resource (document, image or other), as well as its containers, namely, *collection* and *community*. DSpace search therefore is targeted toward retrieval of items only, i.e., search results are always a list of items or collection and community names. In case we have a list of items, every item in the list comes also with its issue date and the names of its authors (Fig. 10).

With semantic search we retrieve a list of instances (instead of DSpace items) that belong to a specific class: the one represented by the query posed. For example, the top-left part of Fig. 11 shows the result list, coming from the execution of a specific semantic query. It is a list of clickable entities that when selected guide the user to the navigation pane about this particular entity (bottom part of Fig. 11).

In Fig. 11, we notice that the selected item has a `dcterms:type` 'Book'. Clicking on 'Book' we now see detailed information regarding 'Book' as an *entity* itself. We also notice that we have *indirectly* retrieved every item that has type 'Book' (through the inverse `dcterms:type` property).

In addition, we find out that 'Book' belongs to the `dspace-ont:DspaceType` class, clicking on which we trigger semantic search to fetch all instances of this class. In this way we return back to the initial semantic search interface, namely, the page where a user can pose queries and obtain class instances. As we can see, retrieved results are not DSpace items. The same naturally holds for other indirect DSpace entities we have reified, such as authors, formats, etc.

5.2.3 Improved search results

Another advantage of semantic search is to allow retrieval of more as well as more precise results than keyword-based search. In order to show this, we comment on and evaluate the results obtained after posing some representative semantic queries.

Suppose first we would like to retrieve all items that contain image files. Searching with the keyword 'image' in the traditional repository search returns two items (Fig. 12).

However, examining each of these items metadata reveals that only the latter has actually an 'image/gif' format. The other has a format of 'application/pdf'; the reason why it is returned by traditional search is that DSpace searches also inside the document's text and happens to meet the word 'image'.

Results 1-4 of 4.

Community Hits:

Community Name
Πολιτισμός και Νέες Τεχνολογίες

Collection hits:

Collection Name
Τεχνολογίες Πληροφορικής & Τηλεπικοινωνιών (Τ.Π.Ε.) και Εκπαίδευση

Item hits:

Issue Date	Title	Author(s)
4-Dec-2006	The value and use of technologies for the ipreservation and promotion of cultural heritage	*Παπαθεοδώρου, Θεόδωρος; Papatheodorou, Theodore*
Nov-2005	Ψηφιοποίηση μνημείων στην Αχαΐα και Αιτωλοακαρνανία	*Καζαντζή, Αθανασία*

Fig. 10 Results list as retrieved through the DSpace keyword-based search

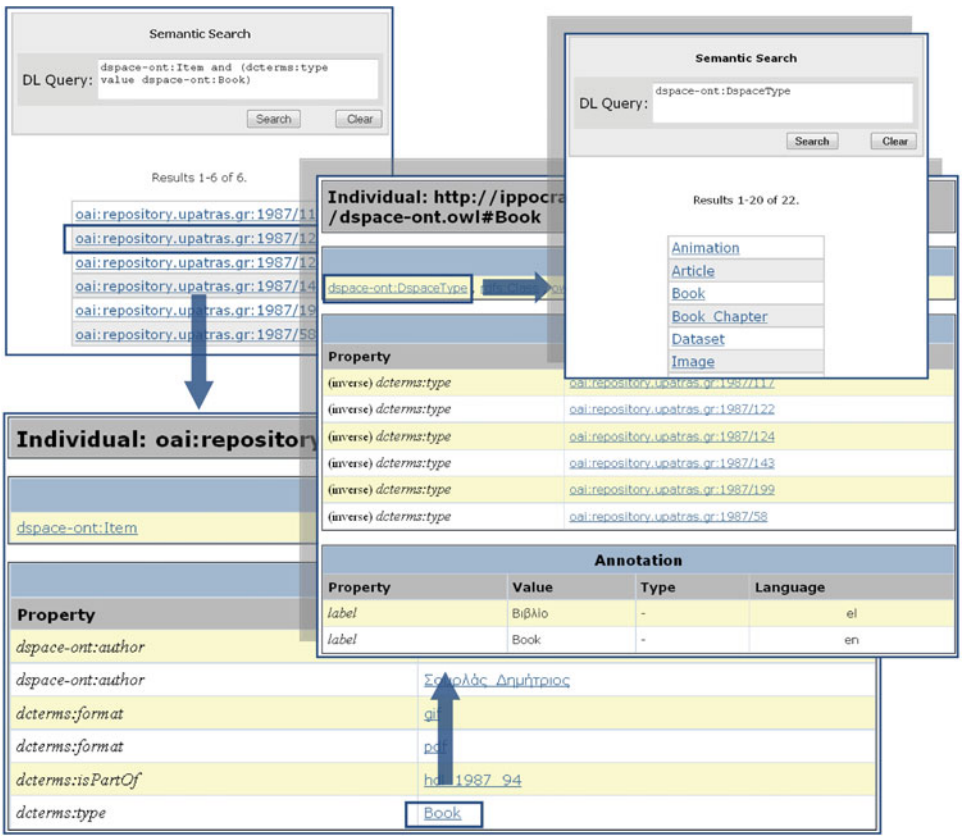


Fig. 11 Entity retrieval using the navigation pane

Search Results

Search: All of Repository

for image Go

Results 1-2 of 2.

Item hits:

Issue Date	Title	Author(s)
3-Mar-2004	Good Practices Handbook	*Dawson, David; Drake, Karl-Magnus*
2005	Συνήθεις Διαφορικές Εξισώσεις	*Σουρλάς, Δημήτριος*

Fig. 12 Simple search in DSpace about the keyword 'image'

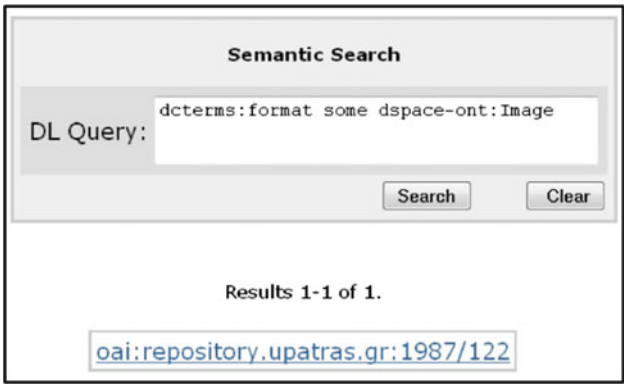


Fig. 13 Semantic search about individuals with 'image' format

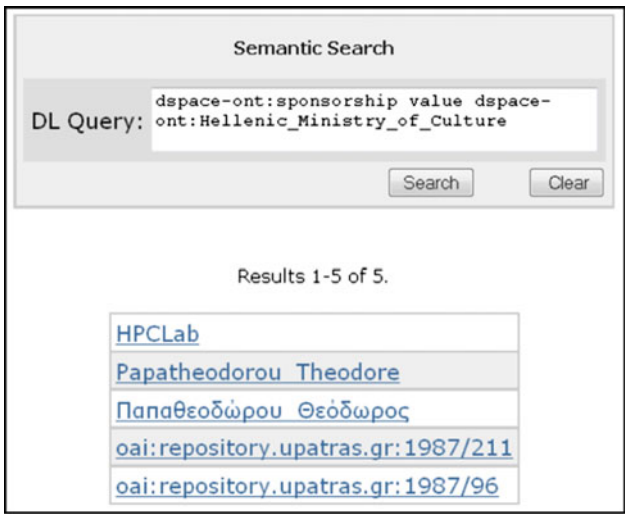


Fig. 15 Semantic search about sponsorships by the 'Hellenic Ministry of Culture'

On the other hand, semantic search fetches just one item, exactly the one that has format `dspace-ont:gif` (see Fig. 13), thus giving the user the chance to obtain more accurate results.

Suppose now we would like to find out who draws sponsorship from a specific institution, for example, what is funded by the 'Hellenic Ministry of Culture' (see Fig. 14). Searching with these keywords through traditional search returns two items that, in their 'sponsorship' metadata field, the above organization has been entered.

Semantic search however retrieves also the authors of these items, aside from the items themselves, thus leading to a total of five results (see Fig. 15).

This is a direct consequence of the role-chain we have declared in our ontology, described in Sect. 3.4 (Semantic Refinement phase, step (d)). In other words, semantic search may also obtain additional results that are logically consequent to the query, thus augmenting the traditional result set.

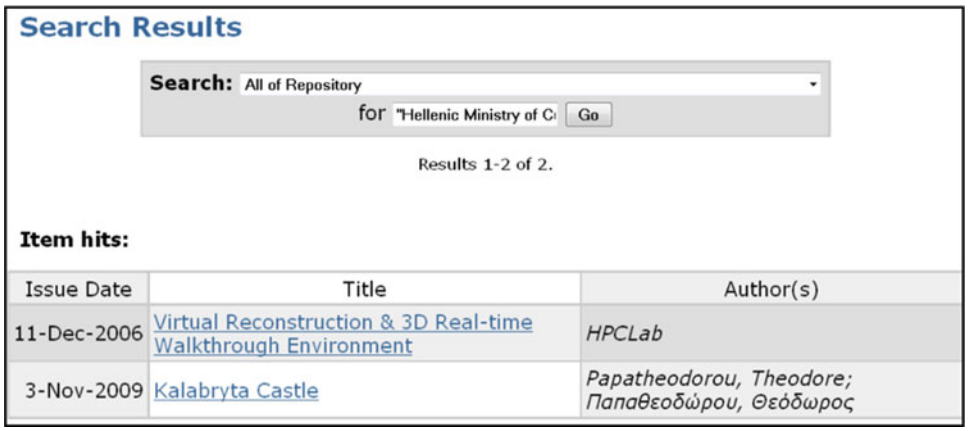
Finally, suppose we would like to retrieve the items that are of type 'Presentation'. The type of items is indexed through the `dcterms:type` metadata field. Searching with 'Presentation' in simple (or advanced) search returns 17 items (Fig. 16). In fact, some may not actually have a 'Presentation' type, possibly due to DSpace's full text indexing discussed earlier.

Now, with semantic search we ask the same question and retrieve 18 items, that is, one more than simple keyword search (Fig. 17). Taking a closer look at the additional item's metadata we notice that the 'Presentation' keyword is not included, thus there is no chance for simple search to retrieve this item, no matter how the underlying indexer is build.

We notice however that the item has a type of `lom:Slide`. Reasonably therefore, it is a presentation, but its author missed to describe it as such. Fortunately though, in our ontology we have modeled this very fact (recall last axiom of Sect. 3.4 (d)): what has a type of `lom:Slide` has also a type of `dspace-ont:Presentation`. The underlying reasoner in semantic search is able to make this inference and as a consequence, the additional item is fetched.

To conclude, the queries above constitute indicative examples that show how the semantic search service can enable a new search and navigation perspective among repository's

Fig. 14 Simple search about 'Hellenic Ministry of Culture'



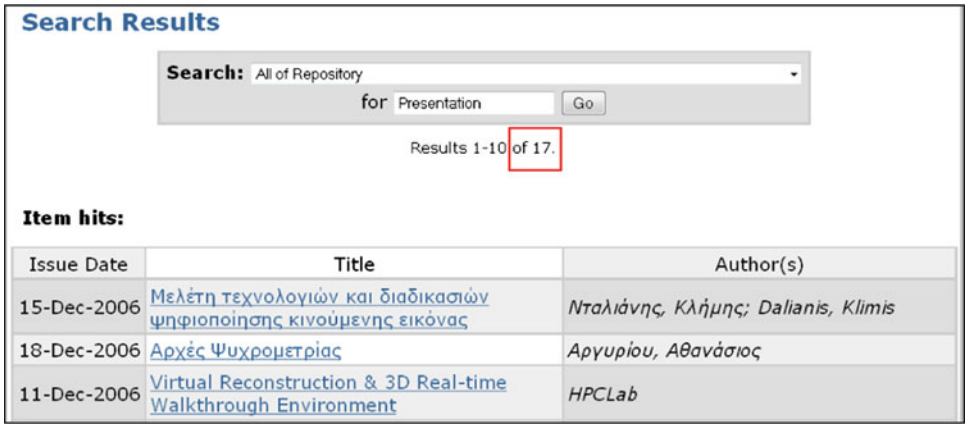


Fig. 16 Keyword-based search for 'Presentation'

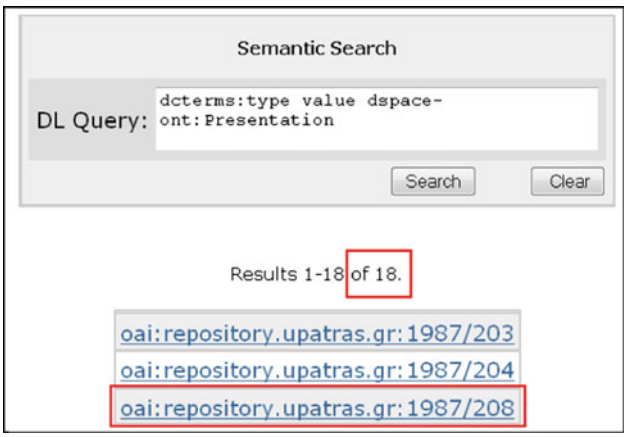


Fig. 17 Searching for instances of type 'Presentation' using the semantic search service

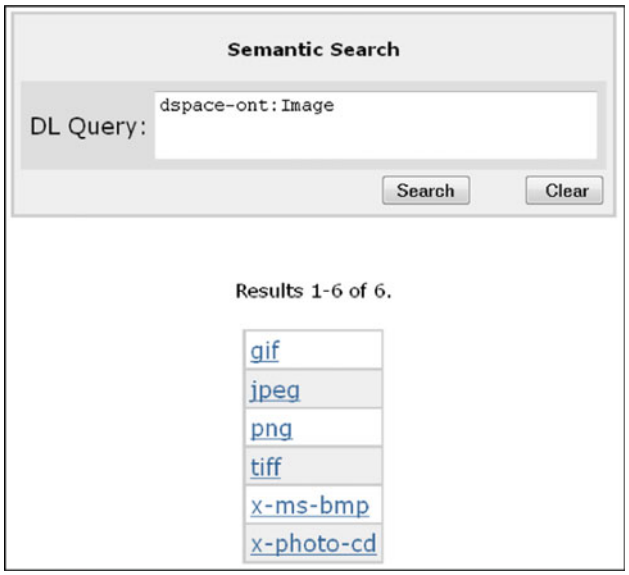


Fig. 18 Semantic search about instances of the 'image' class

contents, by retrieving logically sound and complete results, as per each specific query. This comes at no surprise though; our method is based upon a reasoning system which, in addition, is always guaranteed to supply precise and deterministic results.

5.2.4 Knowledge discovery

In the 'image' example above, semantic search is able to fetch the particular item, despite the fact that its format is declared just as `dspace-ont:gif` (i.e., there is no direct reference to the 'image' keyword). This is because our semantic services are aware that `dspace-ont:gif` is an instance of the `dspace-ont:Image` class and thus they are able to conduct an *inference*; that is, since we ask for a `dspace-ont:Image` format, we also ask for every instance of this class (see Fig. 18).

This knowledge discovery capability can also be determined by asking, for example, for items that are characterized by more than one type. This particular request is impossible to be expressed through traditional search. In the semantic search interface, though, we can pose the following query:

```
dcterms : type min 2 owl : Thing
```

Similarly, with semantic queries we are able to retrieve the co-authors of a particular author. This is possible due to the fact that we have defined the `co_author` property in our ontological model. The corresponding query may have the following form:

```
dspace-ont : co_author some (foaf :
  surname value "Solomou")
```

5.3 Scalability

During our evaluation process we also experimented with other, larger datasets. These datasets were obtained by combining information from other OAI-PMH compliant repositories, resulting in over a thousand records. The size of the produced ontology was *20* MB. The total number of individuals and axioms, which was significantly larger in

Table 5 Statistical information about the test dataset

Metrics	Counts
Individuals	3, 551
Individual axioms	
Class assertion axioms	1, 884
Object property assertion	4, 610
Data property assertion	36, 018

comparison to our repository's ontology, is summarized in Table 5.

It is worth noting that even the ontology creation process, implemented through recursive OAI calls, took a significant amount of time (about 10 *minutes*), already suggesting that this ontology is hardly manageable. Nevertheless, we fed this ontology to our implementation and succeeded in conducting a series of semantic queries, each time receiving sound, complete, and timely results.

As expected, however, the manipulation of this large ontology soon caused the underlying inference engine to crash (this time, FaCT++ v1.3). It is therefore reasonable to conclude that the scalability of our approach is directly dependent on the scalability of the inference engine used.

Consequently, especially when ending up with large-scale ontologies, it would be interesting to see how a persistent store for OWL documents, able to support dynamic ontology updates and incremental reasoning, could be utilized in practice. These techniques, however, especially with respect to OWL DL and OWL 2, are currently beyond the state of the art and form a fruitful field of research upon which we intend to drive our future work.

6 Conclusions

Semantic Web can offer a new and challenging dimension in the way information is managed and manipulated by traditional digital repositories. The representation of metadata as Web ontologies can take advantage of their underlying logical framework: First, this enables the possibility to answer queries based on automated reasoning that would otherwise be impossible; on the other hand, it enriches existing descriptions by revealing or making explicit the implied semantic relations between resources, now regarded as ontological instances.

While it can be costly to produce rich semantic descriptions from scratch, we have shown that we can achieve semantic enrichment of existing flatly described resources in a centralized manner; in such a way the burden on content curators and end users is alleviated and a potential solution to the Semantic Web "chicken-egg" problem [8] is suggested. This process bridges the gap between flatly organized metadata and ontological descriptions, thus achieving a level of *semantic* interoperability. In addition, by relying solely on OAI-PMH and other interoperability standards, this method can be easily extended to accommodate any other digital repository or data provider in general.

We have also shown how to augment traditional digital repository services by implementing an extensible semantic search and navigation facility on top of DSpace. Therefore, the semantic search plug-in consists a noteworthy contribution toward semantic knowledge management in DSpace. In addition, it can have some other practical ramifications on prospective DSpace services, like for example its synergy with authority list control, controlled vocabularies, thesauri support, etc.

Most important though, our facility relies purposely on the OWL API and is designed to be independent of the underlying system, following a "plug-in" philosophy. In combination with the ontology creation and population process, this facility could semantically enable any web-based digital repository system.

Our results confirm that it is possible to navigate among a repository's metadata in more flexible and associative ways. In addition, semantic search can improve traditional keyword search by retrieving additional as well as more semantically accurate results. And of course, semantic search allows the expression of queries that cannot be expressed by simple keyword-based retrieval.

Finally, it can be seen that the use of ontologies in digital repositories and other information systems, in the way it is suggested in this article, can benefit from an ontology harvesting and exchange protocol (just as OAI does for metadata), as well as from a standard and semantics-aware language for querying OWL documents.

References

1. Crofts, N., Doerr, M., Gill, T.: The CIDOC conceptual reference model: a standard for communicating cultural contents. Cultiv. Interact. Issue 9 (on line) (2003). http://www.cultivate-int.org/issue9/chios/
2. DCMI-Libraries Working Group. Library application profile. (2004) http://dublincore.org/documents/2004/09/10/
3. DCMI Usage Board. DCMI metadata terms. (2008) http://dublincore.org/documents/dcmi-terms/
4. Dill, S., Eiron, D., Gibson, D., et al.: SemTag and Seeker: bootstrapping the Semantic Web via automated semantic annotation. In: 12th International conference on world wide web, Budapest, Hungary (2003)
5. Donini, F.: Complexity of reasoning. The description logic handbook: theory, implementation, and applications, pp. 96–136 (2003)
6. Fedora Development Team.: Fedora Open Source Repository Software: White Paper. Fedora Project (2005)
7. Fernandez, M., Lopez, V., Motta, E., Sabou, M., Uren, V., Vallet, D., Castells, P.: Using TREC for cross-comparison between classic

IR and ontology-based search models at a Web scale. In: Semantic search workshop at 18th international world wide web conference, Madrid, Spain (2009)
8. Hendler, J.: Web 3.0: chicken farms on the Semantic Web. Computer **41**(1), 106–108 (2008)
9. Horridge, M., Bechhofer, S., Noppens, O.: Igniting the OWL 1.1 touch paper: the OWL API. In: OWL experiences and directions workshop, Innsbruck, Austria (2007)
10. Horridge, M., Patel-Schneider, P.S.: Manchester Syntax for OWL 1.1. In: OWL experiences and directions workshop (2008)
11. Horrocks, I, Kutz, O., Sattler, U.: The even more irresistible SROIQ. In: 10th international conference on principles of knowledge representation and reasoning, pp. 57–67. AAI Press, San Diego (2006)
12. IEEE LTSC.: Draft standard for learning object metadata (2002). http://ltsc.ieee.org/wg12/files/LOM_1484_12_1_v1_Final_Draft.pdf
13. IEEE LTSC.: Draft recommended practice for expressing IEEE learning object metadata instances using the Dublin Core abstract model (IEEE P1484.12.4/D1) (2008). http://dublincore.org/educationwiki/DCMIIEEELTSCTaskforce?action=AttachFile&do=get&target=LOM-DCAM-newdraft.pdf
14. Koutsomitropoulos, D.A., Tsakou, A.A., Tsolis, D.K., Papatheodorou, T.S.: Towards the development of a general-purpose digital repository. In: 6th international conference on enterprise information systems, Porto, Portugal (2004)
15. Koutsomitropoulos, D., Paloukis, G., Papatheodorou, T.: From metadata application profiles to semantic profiling: ontology refinement and profiling to strengthen inference-based queries on the Semantic Web. Int. J. Metadata Semant. Ontol. **2**(4), 268–280 (2007)
16. Koutsomitropoulos, D., Solomou, G., Papatheodorou, T.: Semantic interoperability of Dublin Core metadata in digital repositories. In: 5th international conference on innovations in information technology, Al Ain, UAE (2008)
17. Koutsomitropoulos, D., Solomou, G., Alexopoulos, A., Papatheodorou, T.: Digital repositories and the Semantic Web: semantic search and navigation for DSpace. In: 4th international conference on open repositories, Atlanta, USA (2009)
18. Kruk, S.R., Decker, S., Zieborak, L.: JeromeDL—reconnecting digital libraries and the Semantic Web. In: 14th international World wide web conference, Chiba, Japan (2005)
19. Lagoze, C., Van de Sompel, H., Nelson, M., Warner, S.: The open archive initiative protocol for metadata harvesting (2002). http://www.openarchives.org/OAI/openarchivesprotocol.html
20. Liebig, T., Luther, M., Noppens, O.: The OWLlink protocol. In: OWL experience and directions workshop, Virginia, USA, 2009
21. McGuiness, D.L., van Harmelen, F.: OWL Web ontology language overview. W3C Recommendation (2004). http://www.w3.org/TR/owl-features/
22. Motik, B., Patel-Schneider, P.F., Parsia, B.: OWL 2 web ontology language: structural specification and functional-style syntax. W3C Recommendation (2009). http://www.w3.org/TR/owl2-syntax/
23. Nilsson, M., Powell, A., Johnston, P., Naeve, A.: Expressing Dublin Core metadata using the resource description framework (RDF). DCMI recommendation (2008). http://dublincore.org/documents/dc-rdf/
24. Nucci, M., Hahn, D., Barbera, M.: The Talia Library Platform—rapidly building a digital library on rails. In: 4th workshop on scripting for the Semantic Web, European Semantic Web conference, Tenerife, Spain (2008)
25. Parsia, B., Patel-Schneider, P.F.: OWL 2 Web ontology language: primer. W3C recommendation (2009). http://www.w3.org/TR/owl2-primer/
26. Powell, A.,Nilsson, M., Naeve, A., Johnston, P., Baker, T.: DCMI abstract model. DCMI Recommendation (2007) http://dublincore.org/documents/abstract-model/
27. Risse, T., Knezevic, P., Meghini, C., Hecht, R., Basile, F.: The BRICKS infrastructure—an overview. In: International conference on electronic information and visual arts, Moscow (2005)
28. Sirin, E., Parsia, B.: SPARQL-DL: SPARQL query for OWL-DL. In: OWL experiences and directions workshop, Innsbruck, Austria (2007)
29. Tsarkov, D., Horrocks, I., Patel-Schneider, P.: Optimizing terminological reasoning for expressive description logics. J. Autom. Reason. **39**(3), 277–316 (2007)