DELIVERABLE

Project Acronym:	DCA
Grant Agreement number:	270927
Project Title:	Digitising Contemporary Art

D3.2 Recommendations on Contextualisation and Enrichment of Contemporary Art

Revision: V1.1

Author(s): Sam Coppens (iMinds/MMLab)
Erik Mannens (iMinds/MMLab)

Project co-funded by the European Commission within the ICT Policy Support Programme		
Dissemination Level		
P	Public	X
C	Confidential, only for members of the consortium and the Commission Services	

1.1.1 REVISION HISTORY AND STATEMENT OF ORIGINALITY

Revision History

Revision	Date	Author	Organisation	Description
0.1	14/12/2012	Sam Coppens / Eric Lammens	iMinds/MMLab	First draft
0.2	17/12/2012	Kathryn Mathe	Central European University	External review
0.3.	19/12/2012	Sam Coppens / Eric Lammens	IBBT/MMLab	Second draft
1.0.	31/12/2012	Rony Vissers	PACKED	Internal review and final version
1.1	17/01/2013	Rony Vissers	PACKED	Appendix proofread by native English speaker

Statement of originality:

This deliverable contains original unpublished work except where clearly indicated otherwise. Acknowledgement of previously published material and of the work of others has been made through appropriate citation, quotation or both.

Content Table

Executive Summary

This deliverable elaborates on applying semantic web technologies in order to contextualise and enrich your information. First, we'll introduce some basic semantic web building blocks:

- data descriptions using RDF and URIs;
- adding semantics using RDFS and OWL;
- querying RDF using SPARQL;
- publishing RDF as Linked Open Data.

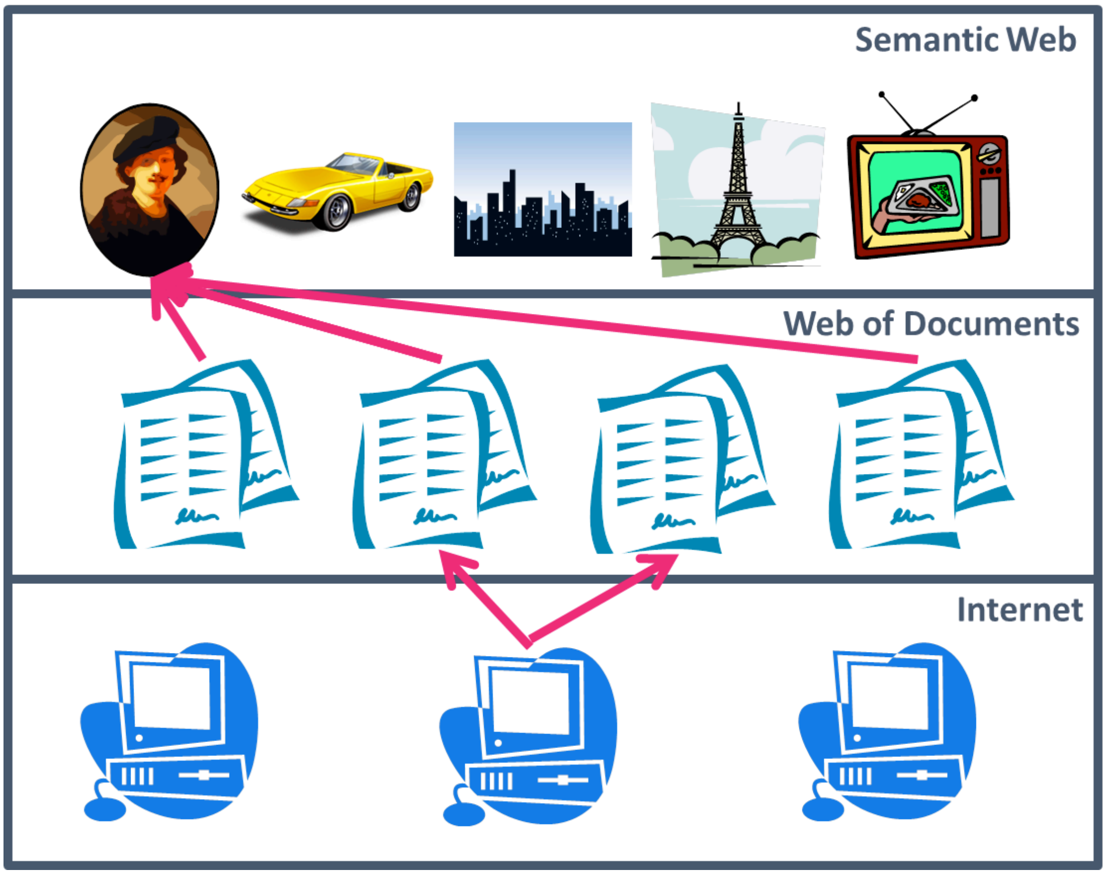
After this introduction, we'll generate some best practices for describing and contextualising data. Contextualisation takes place on several levels. It starts with the data model and ensures the achievement of some interoperability with other data models or even the inheritance of some of their semantics. Then, we'll give some best practices for the data descriptions. The idea here is that the data descriptions must have enough information to describe the data unambiguously. If one is able to achieve this, a good context is created. Finally, we'll discuss how to enrich data with external data sources, to maximise the data contextualisation.

One way of contextualising data is through the use of controlled vocabularies. Special attention goes to the introduction of SKOS for describing controlled vocabularies. Also here, we'll give some best practices on how to model vocabularies. In the last chapter, we'll apply these best practices to the DCA vocabulary, designed to support multilingual search in Europeana.

1 *Semantic Web*

1.1 *Introduction*

The World Wide Web has long been evolving towards the vision of the Semantic Web — an extension of the existing web through which machines are able to understand the data on the Web and, as a consequence, can manage it all. It promises to infuse the Internet with a combination of metadata, structure, and various technologies so that machines can derive meaning from information, make more intelligent choices, and complete tasks with reduced human intervention.



The Semantic Web is an extension of the Web of documents. This means it builds upon existing web architecture to achieve this Semantic Web. Before the Web, we had the Internet. On the Internet, computers were connected to each other through telephone wires. Technologies like the Domain Name System and protocols like Transmission Control Protocol (TCP) and Internet Protocol (IP) allowed computers to communicate to each other. Later on, the Web emerged. This Web was a web of documents, e.g., web sites, which were exchanged between two computers (a Web server and a Web browser). Enabling technologies for this Web were uniform resource identifiers (URIs), HyperText Transfer Protocol (HTTP) and HyperText Markup Language (HTML). Thus from a network of interconnected computers, the Internet, the Web evolved to a network of interconnected documents. The first version of the Web (i.e., Web 1.0) was a read-only Web. People could look up information, but couldn't interact with the information. This was the era of static web sites. Later on, this read-only Web became a read-write Web (i.e., Web 2.0). This stage was characterised by user interaction and collaboration in the form of user generated content (e.g., YouTube) and social networks (e.g., Facebook or Twitter). Currently, the Web is transforming into a Semantic Web (i.e., Web 3.0) where machines can read and understand the underlying data. In this Semantic Web, we don`t describe documents anymore, but things (e.g., a painting of Warhol, Warhol himself, his birthplace Pittsburgh, etc.). When describing these things, the information can of course come from different sources on the Web, e.g., documents, or services. The Semantic Web becomes a web of interconnected Web resources. Enabling technologies for the Semantic Web are URIs, Resource Description Framework (RDF), RDF Schema (RDFS) and Web Ontology Language (OWL), SPARQL Protocol and RDF Query Language (SPARQL), and Linked Data. URIs give things, described on the Web, a name, so they can be referenced. Things identified with a URI are called “resources”. RDF allows information to be modelled as a graph of interconnected resources, identified by URIs. RDFS and OWL add semantics to the RDF graph. SPARQL allows the RDF graph to be queried and finally, Linked Data shows how this data must be published on the Web, in such a way that both machines and humans can understand the data.

1.2 ***URIs: Identify Resources***

Uniform Resource Identifier[1] or URI is a string of characters used to identify a name or a web resource. Such identification enables interaction with representations (e.g., a digital file, a metadata description, etc.) of the resource over a network (typically the World Wide Web) using specific protocols, e.g., HTTP, or File Transfer Protocol (FTP). Anyone can create URIs, and their ownership is clearly delegated, so they form an ideal base technology upon which to build a global Web on top. The syntax of URIs is carefully governed by the IETF, who published RFC 2396 as the general URI specification. The W3C maintains a list of URI schemes.

URIs can be classified as Uniform Resource Locators (URLs), as Uniform Resource Names (URNs), or as both. A URN identifies a resource, e.g., an ISBN number. A URL provides a method for finding this resource, e.g., an HTTP URI. If a resource (a thing with a URI) is resolvable on the Web (meaning the URI is also a URL), we talk about web resources. These are things identified by a URI and a location (URL) to retrieve it.

1.3 ***RDF: Describe Resources***

In the Semantic Web, information is represented using RDF (1). The underlying structure of RDF data is a collection of triples. Every triple consists of a subject (S), predicate or property (P) and object (O) respectively. A set of such triples is called an RDF graph.

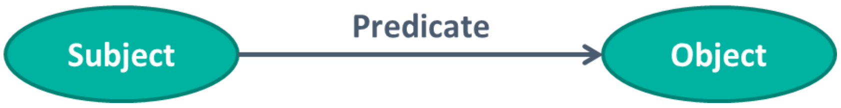


Figure 1: Triple with a URI as object

[1] http://tools.ietf.org/html/rfc3986

Note that the object of a triple can also be a literal instead of a URI. However, literals cannot appear as the subject or predicate of a triple. By convention, a literal is represented using a box instead of an ellipse.

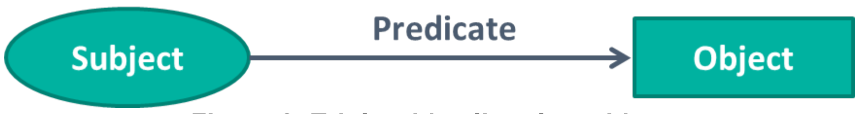


Figure 2: Triple with a literal as object

All literals have a lexical form being a Unicode string. A literal can also be provided with a language tag. Alternatively, a datatype URI can be provided to a literal, forming a typed literal. If we take the example of the record of Damien Hirst's artwork 'For the Love of God' record and put it in RDF, the record becomes a graph, represented below.

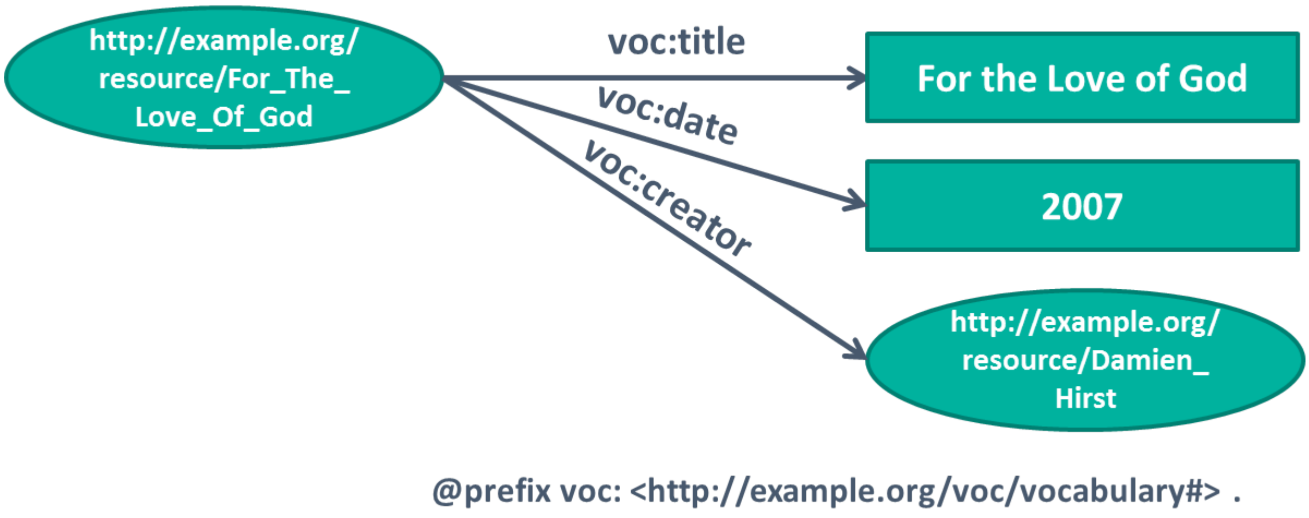

http://example.org/resource/For_The_Love_Of_God
voc:title
voc:date
voc:creator
For the Love of God
2007
http://example.org/resource/Damien_Hirst
@prefix voc: <http://example.org/voc/vocabulary#> .

Figure 3: Example of graph describing an artwork by Damien Hirst

1.3.1 RDF serialisation

RDF transforms data into a data graph. This graph can be serialised into a document for data exchange. Serialisation is thus nothing more than the rendering of a textual representation for data exchange. As explained earlier, the Semantic Web builds upon the Web of documents, where the exchange of documents stands central. When RDF can become serialised into a document, the existing web architecture can be exploited.

W3C have developed an Extensible Markup Language (XML) RDF serialisation in the RDF Model and Syntax recommendation (2). RDF/XML is considered to be the standard interchange format for RDF on the Semantic Web. The example, shown below, serialises the RDF graph example of Figure 3 to RDF/XML.

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:voc ="http://example.org/voc/vocabulary#">
 <rdf:Description rdf:about="http://example.org/resource/For_The_Love_Of_God">
   <rdf:type rdf:resource="http://example.org/voc/vocabulary#Artwork"/>
   <voc:title>For the Love of God</dc:title>
   <voc:description>For the Love of God is a sculpture by artist Damien Hirst
            produced in 2007. It consists of a platinum cast of a human skull
            encrusted with 8,601 flawless diamonds, including a pear-shaped pink
            diamond located in the forehead.</dc:description>
   <voc:date>2007-06-01T00:00:00</dc:date>
   <voc:creator rdf:resource="http://dbpedia.org/resource/Damien_Hirst"/>
 </rdf:Description>
</rdf:RDF>
```

RDF/XML is not the only serialisation of RDF to a document. There exist several other serialisations. For example, Notation3 (N3, (3)) is an excellent plain text alternative serialisation. N3 is a shorthand non-XML serialisation of RDF models, designed with human-readability in mind. N3 is much more compact and readable than RDF/XML notation. The following figure shows the RDF graph of Figure 3 serialised in N3. N3 has several features that go beyond the serialisation for RDF models, such as support for RDF-based rules.

```
@prefix voc: <http://example.org/voc/vocabulary#>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
<http://example.org/resource/For_The_Love_Of_God>
        rdf:type         voc:Artwork;
        voc:creator      <http://example.org/resource/Damien_Hirst>
        voc:date         "2007-06-01T00:00:00";
        voc:description  "For the Love of God is a sculpture by artist
                         Damien Hirst produced in 2007. It consists of
                         a platinum cast of a human skull encrusted with
                         8,601 flawless diamonds, including a pear-shaped
                         pink diamond located in the forehead.";
        voc:title        "For the Love of God".
```

N-Triples (4) is another way of serialising RDF graphs. It just enumerates all the triples of the RDF graph. It was designed to be a simpler format than Notation 3, and therefore easier for software to parse and generate. This figure shows the RDF graph of Figure 3 serialised in N-Triples.

```
<http://example.org/resource/For_The_Love_Of_God>
    <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
                                                  <http://example.org/voc/
                                                  vocabulary#Artwork">.
<http://example.org/resource/For_The_Love_Of_God>
    <http://example.org/voc/vocabulary#title>     "For the Love of God".
<http://example.org/resource/For_The_Love_Of_God>
    <http://example.org/voc/vocabulary#description> "For the Love of God is a
                                                  sculpture by artist Damien
                                                  Hirst produced in 2007. It
                                                  consists of a platinum cast
                                                  of a human skull encrusted
                                                  with 8,601 flawless diamonds,
                                                  including a pear-shaped pink
                                                  diamond located in the
                                                  forehead.".
<http://example.org/resource/For_The_Love_Of_God>
    <http://example.org/voc/vocabulary#date>      "2007-06-01T00:00:00".
<http://example.org/resource/For_The_Love_Of_God>
    <http://example.org/voc/vocabulary#creator>   <http://example.org/resource/
                                                  Damien_Hirst>.
```

1.4 *Linked Open Data: Publish RDF*

The main goal of Linked Open Data (LOD) (5) is to let people share structured data on the Web as easily as they share documents today. It actually refers to a style of publishing and interlinking structured data on the Web. The main recipe for LOD is RDF (1). The structured data is published as RDF data and RDF links are used to link data from different data sources. This way, the LOD on the Web creates a giant global graph, in which all the published data is inter-connected. In this Web of Data, clients can easily discover and consume data.

LOD stipulates four basic principles [2]:

- The first principle is that to start off we have to identify the items of interest in our domain. Those items are the resources, which will be described in the data.
- The second principle is that those resources have to be identified by HTTP URIs.
- The third principle is to provide useful information when accessing an HTTP URI.
- The fourth principle is to provide links to the outside world, i.e., to connect the data with that of other datasets in the Web of Data. This makes it possible to browse data from a certain server and receive information from another server. In other words, by linking the data with that of other datasets, the web becomes one huge database (the Web of Data).

In practice, this means that all resources should be identified by HTTP URIs. When users visit this HTTP URI, the server should provide the user with an appropriate data format. This means HTML for browsers, RDF for other machine agents. This is done by content negotiation including RDF – in – attributes (RDFa, (6)) in HTML pages.

The enrichment of the data will actually interlink it with data from other data sources. These enrichments will link the data graph to the giant global data graph. In the latter data can easily be discovered and consumed. The figure below shows what the giant global data graph[3] currently looks like. Every dot in the graph is a dataset being published as LOD. The interconnections to the LOD datasets are enrichments.

For those interested in Linked Open Data, a DCA milestone deliverable with very practical guidelines for publishing Linked Open Data, which wasn't public, is included as an appendix to this deliverable.

[2] http://www.w3.org/DesignIssues/LinkedData.html
[3] http://linkeddata.org/

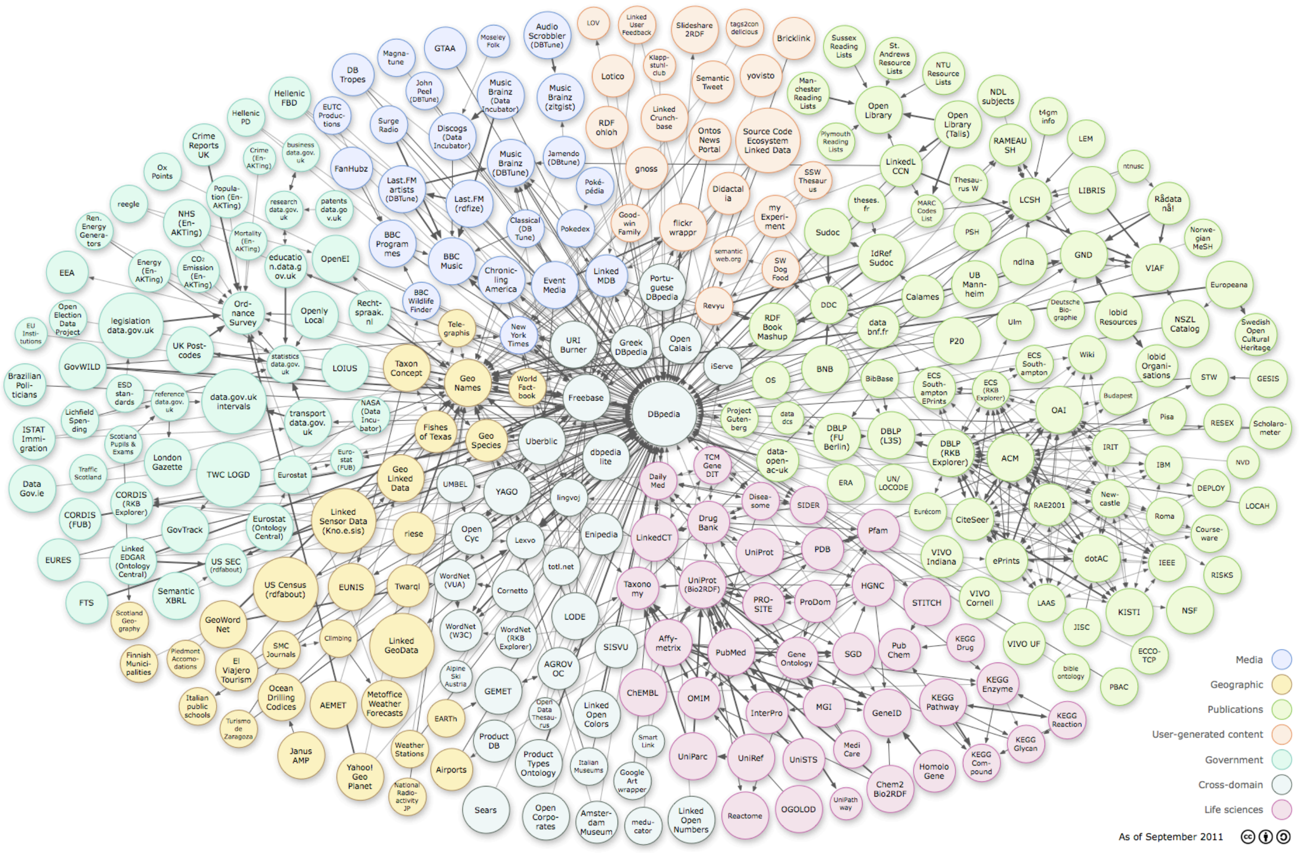


Figure 4: giant global data graph

The enrichments serve a dual purpose. Firstly, they help to disambiguate the data. Secondly, interlinking data brings in some extra information about the resource. The more the data is interlinked with external resources, the more value the data gets and the more meaningful it becomes. By enriching the data, more context information is added to it.

1.5 *SPARQL: Query RDF*

SPARQL (7) was designed to make sure that the content it publishes could be queried. SPARQL is a query language and data access protocol for the Semantic Web. SPARQL is defined in terms of the W3C's RDF data model and will work for any data source that can be mapped into RDF.

Providing a SPARQL endpoint is very important. It opens up the data and lets other data sources use it to enrich their own. SPARQL follows a well-trodden path, offering a simple, reasonably familiar (to SQL users) SELECT query form. The SPARQL query below returns all titles and descriptions of every artwork known by the system. Notice the similarity with SQL, except for the WHERE clauses, which are formulated using RDF.

```
PREFIX voc: <http://example.org/voc/vocabulary#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns>

SELECT ?title ?description
WHERE
{
      ?artwork a voc:Artwork.
      ?artwork voc:title ?title.
      ?artwork voc:description ?description.
}
```

The query can also contain constant values as shown by the query below. In this query we ask for all artworks by Damien Hirst known by the system.

```
PREFIX voc: <http://example.org/voc/vocabulary#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns>

SELECT ?artwork
WHERE
{
      ?artwork a voc:Artwork.
      ?artwork voc:creator <http://example.org/resource/Damien_Hirst>.
}
```

SPARQL queries can also handle FILTER expressions on query variables. In the following query, we ask for all of Damien Hirst's artworks made after 2005.

```
PREFIX voc: <http://example.org/voc/vocabulary#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns>

SELECT ?artwork
WHERE
{
      ?artwork a voc:Artwork.
      ?artwork voc:creator <http://example.org/resource/Damien_Hirst>.
      ?artwork voc:date ?date.
      FILTER (?date > "2005-01-01T00:00:00" ^^xsd:dateTime)
}
```

SPARQL also allows one to do some Boolean operations on the where clauses. To demonstrate this, we'll show a query that will ask for all the artworks by Damien Hirst and Jean-Michel Basquiat.

```
PREFIX voc: <http://example.org/voc/vocabulary#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns>

SELECT ?artwork
WHERE
{
      {
      ?artwork a voc:Artwork.
      ?artwork voc:creator <http://example.org/resource/Damien_Hirst>.
      }
      UNION
      {
      ?artwork a voc:Artwork.
      ?artwork voc:creator <http://example.org/resource/Jean-Michel_Basquiat>.
      }
}
```

1.6 *RDFS / OWL: Add Semantics*

So far, we have introduced RDF, or how data can be modelled as a graph representation using RDF. RDF provides a way to state facts or make assertions about resources, using named properties and values. Users also need to be able to define a vocabulary to use in these statements. RDF properties may be thought of as attributes of resources and in this sense correspond to traditional attribute-value pairs. RDF properties also represent relationships between resources. RDF however, provides no mechanisms for describing such properties, nor does it provide any mechanisms for describing the relationships between such properties and other resources. As such, RDF does not add any semantics to the data.

Semantic models are used to attribute RDF data models with semantics. These models define how one's data is described. RDF data can be encoded with semantic metadata using two syntaxes: RDFS and OWL. While RDF is a graph-based model, RDFS (8) and OWL (9) are object-oriented. Both RDFS and OWL are W3C specifications.

1.6.1 RDFS

RDF's vocabulary description language, RDF Schema, is a semantic extension of RDF. RDFS is defined in the form of an RDF vocabulary. Thus, RDFS descriptions are written in RDF. It is meant to be a simple datatyping model for RDF. Datatyping means that the resources in the RDF graph will have a certain class (type) appended to it, to characterise the resource (e.g., xsd:String, xsd:int, rdfs:Class, etc.). RDFS defines classes and properties that may be used to describe classes, properties and other resources. It provides mechanisms for describing groups of related resources and the relationships between them. These resources are used to determine characteristics of other resources, such as the domains and ranges of properties, subclasses and subproperties. The namespace for RDFS is <http://www.w3.org/2000/01/rdf-schema\#>. The namespace for RDF elements is <http://www.w3.org/1999/02/22-rdf-syntax-ns\#>.

The most important classes RDFS introduces are rdfs:Resource, rdfs:Class and rdf:Property. Next to these classes, RDFS defines some classes for datatyping (e.g., rdfs:Literal) and constructs for representing containers (e.g., rdfs:Container) and RDF statements (e.g., rdf:subject).

Coming back to the example of Figure 3, one could define a class of artists and a class of artworks. These classes can then be used for datatyping the RDF model.

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix voc: <http://example.org/voc/vocabulary#> .
voc:Artist  rdf:type       rdfs:Class.
voc:Artwork rdf:type       rdfs:Class.
```

One could say, for instance, that the resource <http://example.org/resource/Damien_Hirst> is an individual belonging to this class. For this, RDFS introduces the property rdf:type. This property denotes to which class a resource belongs. One can add extra triples to the RDF graph to denote which resource belongs to which class. The statements below shows which triples are added to our RDF graph shown in Figure 3 to datatype the following resources: <http://example.org/resource/Damien_Hirst> and <http://example.org/resource/For_The_Love_Of_God>.

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix voc: <http://example.org/voc/vocabulary#> .
<http://example.org/resource/Damien_Hirst>        rdf:type voc:Artist.
<http://example.org/resource/For_The_Love_Of_God> rdf:type voc:Artist.
```

For properties, RDFS allows one to define their domain and range. In our RDF model example, shown in Figure 3, one could define the domain and range of the properties voc:title, voc:description, voc:date, and voc:creator.

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix voc: <http://example.org/voc/vocabulary#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.
voc:title       rdf:type      rdf:Property;
                rdfs:domain   voc:Artwork;
                rdfs:range    rdfs:Literal.
voc:description rdf:type      rdf:Property;
                rdfs:domain   voc:Artwork;
                rdfs:range    rdfs:Literal.
voc:date        rdf:type      rdf:Property;
                rdfs:domain   voc:Artwork;
                rdfs:range    xsd:dateTime.
voc:creator     rdf:type      rdf:Property;
                rdfs:domain   voc:Artwork;
                rdfs:range    voc:Artist.
```

Alongside RDFS also allows one to define subclasses and subproperties. In our example, we can define a class voc:ContemporaryArtist as a subclass of voc:Artist and a class voc:ContemporaryArtwork as a subclass of voc:Artwork. Because the property voc:creator is defined to have a voc:Artwork as domain and a voc:Artist as range, this property is still usable for linking a voc:ContemporaryArtwork to a voc:ContemporaryArtist. The model shown below shows how to achieve this subclassing. Of course, before subclassing these classes, the two new classes (voc:ContemporaryArtwork and voc:ContemporaryArtist) need to be declared.

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix voc: <http://example.org/voc/vocabulary#> .
voc:ContemporaryArtwork       rdf:type        rdf:Class;
                              rdfs:subClassOf voc:Artwork.
voc:ContemporaryArtist        rdf:type        rdf:Class;
                              rdfs:subClassOf voc:Artist.
```

If we put everything together, we get our first RDFS model, shown below. It is a good practice to always add labels and comments to class and property definitions, because they contribute to the semantics of the class or property. It is important to make a distinction between this RDFS model and the RDF model. The explicit datatyping happens in the RDF model, the RDFS model declares the datatypes and their semantics. For this reason, the triples doing the datatyping belong to the RDF model and not the RDFS model.

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix voc: <http://example.org/voc/vocabulary#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.

voc:Artist                    rdf:type        rdfs:Class;
                              rdfs:label      "artist";
                              rdfs:comment    "This class is used for declaring
                                              artists in your RDF model.".

voc:Artwork                   rdf:type        rdfs:Class;
                              rdfs:label      "artwork";
                              rdfs:comment    "This class is used for declaring
                                              artworks in your RDF model.".

voc:ContemporaryArtwork       rdf:type        rdf:Class;
                              rdfs:label      "contemporary artwork";
                              rdfs:comment    "This class describes a
                                              contemporary artwork";
                              rdfs:subClassOf voc:Artwork.

voc:ContemporaryArtist        rdf:type        rdf:Class;
                              rdfs:label      "contemporary artist";
                              rdfs:comment    "This class describes a
                                              contemporary artist".
                              rdfs:subClassOf voc:Artist.

voc:title                     rdf:type        rdf:Property;
                              rdfs:label      "title";
                              rdfs:comment    "This property gives the title of
                                              an artwork";
                              rdfs:domain     voc:Artwork;
                              rdfs:range      rdfs:Literal.

voc:description               rdf:type        rdf:Property;
                              rdfs:label      "description";
                              rdfs:comment    "This property gives the
                                              description of an artwork";
                              rdfs:domain     voc:Artwork;
                              rdfs:range      rdfs:Literal.

voc:date                      rdf:type        rdf:Property;
                              rdfs:label      "date";
                              rdfs:comment    "This property gives the date
                                              the artwork was created";
                              rdfs:domain     voc:Artwork;
                              rdfs:range      xsd:dateTime.

voc:creator                   rdf:type        rdf:Property;
                              rdfs:label      "creator";
                              rdfs:comment    "This property links an artist
                                              to the artwork as a creator";
                              rdfs:domain     voc:Artwork;
                              rdfs:range      voc:Artist.
```

1.6.2 OWL

OWL can be used to explicitly represent the meaning of terms in vocabularies and the relationships between such terms. This representation of terms and their interrelationships is called an ontology. OWL has more facilities for expressing meaning and semantics than XML, RDF, and RDFS, and thus OWL goes beyond these languages in its ability to represent machine interpretable content on the Web. OWL builds further on RDFS and, as a consequence, is compatible with RDFS, but it allows more expressivity to describe things and to add semantics to descriptions. OWL provides more vocabulary for describing properties and classes: amongst others, relations between classes (e.g. disjointness), cardinality (e.g. "exactly one"), equality, richer typing. Currently, there are two OWL specifications:

- OWL1[4]
- OWL2[5].

OWL2 depreciates OWL1, but is compatible with OWL1. To explain the concepts of OWL, we will start with OWL1. Later on, we will expand on OWL2.

OWL1 foresees a number of constructs to model one's data with more expressivity than RDFS. There exist several kinds of constructs:

- RDFS features:
  - rdfs:Class;
  - rdfs:subClassOf;
  - rdf:Property;
  - rdfs:subPropertyOf;
  - rdfs:domain;
  - rdfs:range;
  - individuals:members of a certain class.

- equality:
  - owl:equivalentClass: to denote that a certain class is the same as another class;
  - owl:equivalentProperty: to denote a certain property means the same as another property;
  - owl:sameAs: to denote certain individuals are actually the same;
  - owl:differentFrom: to express that two individuals are different;
  - owl:AllDifferent: which denotes that a number of individuals are mutually distinct;
  - owl:disjointWith: classes may be stated to be disjoint from each other, meaning that an individual can't at the same time be an instance of two classes which are disjoint.

- property characteristics:
  - owl:inverseOf: expresses that two properties are the inverse of each other;
  - owl:TransivityProperty: a class for transitive properties (A→B and B→C THEN A → C);

[4] http://www.w3.org/TR/owl-features/
[5] http://www.w3.org/TR/owl2-overview

    - owl:SymmetricProperty: a class for symmetrical properties (A → B THEN B → A);
    - owl:FunctionalProperty: if a property is an owl:FunctionalProperty, then it has no more than one value for each individual;
    - owl:InverseFunctionalProperty: if a property is inverse functional, then the inverse of the property is functional.

- property restrictions:
    - owl:allValuesFrom: means that this property on this particular class has a local range restriction associated with it;
    - owl:someValuesFrom: a restriction on a property of which one of its values is of a certain type;
    - owl:hasValue: a property can be required to have a certain individual as a value.

- restricted cardinality:
    - owl:minCardinality: denotes the number of different values a certain property may have with respect to a certain class, e.g., an artwork has at least one creator;
    - owl:maxCardinality: expresses the maximum number of different values a certain property may have with respect to a certain class;.
    - owl:cardinality: combines the two previous cardinality restrictions.

- class intersection:
    - owl:intersectionOf: defines a class as the intersection of two classes, e.g., one can define the class of employed persons as the intersection of a person class and an employed things class;
    - owl:unionOf: defines a class as the union of two classes;
    - owl:complementOf: defines a class as the complement of another class. e.g., females and males.

- class enumeration:
    - owl:oneOf: classes can be described by enumeration of the individuals that make up the class. The members of the class are exactly the set of enumerated individuals; not more, not less.

OWL1 provides three increasingly expressive sub-languages designed for use by specific communities of implementers and users:

- OWL Lite: OWL Lite supports those users primarily needing a classification hierarchy and simple constraints. OWL Lite provides a quick migration path for thesauri and other taxonomies. Owl Lite also has a lower formal complexity than OWL DL.
- OWl DL: OWL DL supports those users who want the maximum expressiveness while retaining computational completeness and decidability. OWL DL includes all OWL language constructs, but can only be used under certain restrictions.
- OWL Full: OWL Full is meant for users who want maximum expressiveness and the syntactic freedom of RDF with no computational guarantees. It is unlikely that any reasoning software will be able to support complete reasoning for every feature of OWL Full.

OWL Lite uses only some of the OWL language features and has more limitations on the use of the features than OWL DL or OWL Full. OWL Lite does not use the following features: owl:oneOf, owl:disjointWith, owl:unionOf, owl:complementOf, and owl:hasValue. Furthermore, in OWL Lite classes can only be defined in terms of named superclasses (superclasses cannot be arbitrary expressions), and only certain kinds of class restrictions can be used. For instance equivalence between classes and subclass relationships between classes are also only allowed between named classes, and not between arbitrary class expressions. The same holds true for owl:intersectionOf. The properties for denoting the cardinality in OWL, are also restricted to 0 and 1 in OWL Lite.

Both OWL DL and OWL Full use the same vocabulary. They include owl:oneOf, owl:disjointWith, owl:unionOf, owl:complementOf, and owl:hasValue, which OWL LITE does not use. However, OWL DL is subject to some restrictions. Roughly, OWL DL requires type separation; a class can't also be an individual or property, a property can't also be an individual or class. This implies that restrictions cannot be applied to the language elements of OWL itself (something that is allowed in OWL Full). Furthermore, OWL DL requires that properties are either ObjectProperties or DatatypeProperties. DatatypeProperties are relations between instances of classes and RDF literals and XML Schema datatypes, while ObjectProperties are relations between instances of two classes.

OWL Full can be viewed as an extension of RDF, while OWL Lite and OWL DL can be viewed as extensions of a restricted view of RDF. Every OWL (Lite, DL, Full) document is an RDF document, and every RDF document is an OWL Full document, but only some RDF documents will be a legal OWL Lite or OWL DL document.

OWL2 adds new functionality with respect to OWL1. Some of the new features are syntactic sugar (e.g., disjoint union of classes) while others offer new expressivity, including:

- keys;
- property chains;
- richer datatypes, data ranges;
- qualified cardinality restrictions;
- asymmetric, reflexive, and disjoint properties;
- enhanced annotation capabilities.

In addition, some of the restrictions applicable to OWL DL have been relaxed; as a result, the set of RDF Graphs that can be handled by Description Logics reasoners is slightly larger in OWL2. OWL 2 also defines three new profiles, instead of OWL Lite, OWL DL, and OWL Full:

- OWL 2 EL is particularly useful in applications employing ontologies that contain very large numbers of properties and/or classes. This profile captures the expressive power used by much ontology and is a subset of OWL 2. Dedicated reasoning algorithms for this profile are available and have been demonstrated to be implementable in a highly scalable way.
- \OWL 2 QL is designed so that data that is stored in a standard relational database system can be queried through an ontology via a simple rewriting mechanism, i.e., by rewriting the query into an SQL query that is then answered by the RDBMS system, without any changes to the data. Thus, reasoning on these ontologies could be performed by rewriting incoming queries.

- OWL 2 RL is aimed at applications that require scalable reasoning without sacrificing too much expressive power. OWL 2 RL reasoning systems can be implemented using rule-based reasoning engines. Such ontologies can be transformed into rules to perform the reasoning tasks.

Let's assume, we want to build an ontology for the RDF graph shown in Figure 3. We start by deciding to build an OWL ontology for it. Next, we identify the classes. We decide we want to build the following classes: Artwork, Artist, ContemporaryArtwork, ContemporaryArtist. We can create them right away as shown by the following statements:

```
:Artwork rdf:type owl:Class .
:ContemporaryArtwork rdf:type owl:Class .
:Artist rdf:type owl:Class .
:ContemporaryArtist rdf:type owl:Class .
```

The next thing we do is define the properties for each class. It is obvious that Artwork and ContemporaryArtwork will have the same properties, as well as Artist and ContemporaryArtist. In fact, ContemporaryArtwork will become a subclass of Artwork and ContemporaryArtist a subclass of Artist. Subclasses inherit the properties of their superclasses. An Artwork will be described by a creator, a creation date, a title, and possibly a description. Artists will, of course, be the creator of artworks. In this example, we focus on describing an artwork and on their properties. We can create the properties as shown below:

```
:created rdf:type owl:ObjectProperty ;
         rdfs:domain :Artist ;
         rdfs:range :Artwork ;
         owl:inverseOf :creator .

:creator rdf:type owl:ObjectProperty ;
         rdfs:range :Artist ;
         rdfs:domain :Artwork .

:date rdf:type owl:DatatypeProperty ;
      rdfs:domain :Artwork ;
      rdfs:range xsd:dateTime .

:description rdf:type owl:DatatypeProperty ;
             rdfs:domain :Artwork ;
             rdfs:range xsd:string .

:title rdf:type owl:DatatypeProperty ;
       rdfs:domain :Artwork ;
       rdfs:range xsd:string .
```

We have immediately defined the domains and ranges of the properties. We also have created the property 'created' as the inverse of the property 'creator'. Now, we can improve the definition of our classes.

Thus, an artwork has:
- exactly one creation date;
- minimum one artist who created it;
- minimum one title.

```
:Artwork rdf:type owl:Class ;
```

```
         rdfs:subClassOf [ rdf:type owl:Restriction ;
                           owl:onProperty :date ;
                           owl:qualifiedCardinality "1"^^xsd:nonNegativeInteger ;
                           owl:onDataRange xsd:string
                         ] ,
                         [ rdf:type owl:Restriction ;
                           owl:onProperty :creator ;
                           owl:onClass :Artist ;
                           owl:minQualifiedCardinality "1"^^xsd:nonNegativeInteger
                         ] ,
                         [ rdf:type owl:Restriction ;
                           owl:onProperty :title ;
                           owl:minQualifiedCardinality "1"^^xsd:nonNegativeInteger ;
                           owl:onDataRange xsd:string
                         ] .
```

An artist is defined only as having created minimum one artwork.

```
:Artist rdf:type owl:Class ;
        rdfs:subClassOf [ rdf:type owl:Restriction ;
                          owl:onProperty :created ;
                          owl:onClass :Artwork ;
                          owl:minQualifiedCardinality "1"^^xsd:nonNegativeInteger
                        ] .
```

A contemporary artwork is a subclass of artworks in general. It also inherits their restrictions (exactly one creation date, etc.), but has the extra restriction that its creation date must be more recent than 1945.

```
:ContemporaryArtwork rdf:type owl:Class ;
                     rdfs:subClassOf :Artwork ,
                                     [ rdf:type owl:Restriction ;
                                       owl:onProperty :date ;
                                       owl:someValuesFrom [ rdf:type rdfs:Datatype ;
                                                            owl:onDatatype xsd:dateTime ;
                                                            owl:withRestrictions
                                                                   (   [   xsd:minExclusive
                                                                   "1945-01-01T00:00:00"
                                                                       ]
                                                                   )
                                                          ]
                                     ] .
```

A contemporary artist is defined as having created at least one contemporary artwork.

```
:ContemporaryArtist rdf:type owl:Class ;
                    rdfs:subClassOf :Artist ,
                                    [ rdf:type owl:Restriction ;
                                      owl:onProperty :created ;
                                      owl:onClass :ContemporaryArtwork ;
                                        owl:minQualifiedCardinality
                                        "1"^^xsd:nonNegativeInteger
                                    ] .
```

The last thing we do is interrelate our classes and properties with those of other ontologies to promote interoperability. An artist can be defined as a subclass of foaf:Person. The properties date, title, description, and creator can be made subproperties of some Dublin Core properties, respectively, dc:date, dc:title, dc:description, and dc:creator. This is done using the following statements:

```
:Artist rdfs:subClassOf foaf:Person.
:creator rdfs:subPropertyOf dc:creator.
```

```
:date rdfs:subPropertyOf dc:date.
:title rdfs:subPropertyOf dc:title.
:description rdfs:subPropertyOf dc:description.
```

If we put everything together we get the following ontology:

```
@prefix : <http://example.org/voc/vocabulary#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix dc: <http://purl.org/dc/elements/1.1/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix voc: <http://example.org/voc/vocabulary#> .
@prefix xml: <http://www.w3.org/XML/1998/namespace> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@base <http://example.org/voc/vocabulary> .

<http://example.org/voc/vocabulary> rdf:type owl:Ontology .

:Artwork rdf:type owl:Class ;
         rdfs:subClassOf [ rdf:type owl:Restriction ;
                           owl:onProperty :date ;
                           owl:qualifiedCardinality "1"^^xsd:nonNegativeInteger ;
                           owl:onDataRange xsd:string
                         ] ,
                         [ rdf:type owl:Restriction ;
                           owl:onProperty :creator ;
                           owl:onClass :Artist ;
                           owl:minQualifiedCardinality "1"^^xsd:nonNegativeInteger
                         ] ,
                         [ rdf:type owl:Restriction ;
                           owl:onProperty :title ;
                           owl:minQualifiedCardinality "1"^^xsd:nonNegativeInteger ;
                           owl:onDataRange xsd:string
                         ] .

:Artist rdf:type owl:Class ;
        rdfs:subClassOf foaf:Person ,
                                      [ rdf:type owl:Restriction ;
                          owl:onProperty :created ;
                          owl:onClass :Artwork ;
                          owl:minQualifiedCardinality "1"^^xsd:nonNegativeInteger
                        ] .

:ContemporaryArtwork rdf:type owl:Class ;
                     rdfs:subClassOf :Artwork ,
                                     [ rdf:type owl:Restriction ;
                                       owl:onProperty :date ;
                                       owl:someValuesFrom [ rdf:type rdfs:Datatype ;
                                                            owl:onDatatype xsd:dateTime ;
                                                            owl:withRestrictions    (    [
xsd:minExclusive "1945-01-01T00:00:00"
                                                                                     ]
                                                                                   )
                                                          ]
                                     ] .

:ContemporaryArtist rdf:type owl:Class ;
                    rdfs:subClassOf :Artist ,
                                    [ rdf:type owl:Restriction ;
                                      owl:onProperty :created ;
                                      owl:onClass :ContemporaryArtwork ;
                                      owl:minQualifiedCardinality
"1"^^xsd:nonNegativeInteger
                                    ] .
```

```
:created rdf:type owl:ObjectProperty ;
         rdfs:domain :Artist ;
         rdfs:range :Artwork ;
         owl:inverseOf :creator .

:creator rdf:type owl:ObjectProperty ;
        rdfs:subPropertyOf dc:creator;
         rdfs:range :Artist ;
         rdfs:domain :Artwork .

:date rdf:type owl:DatatypeProperty ;
      rdfs:subPropertyOf dc:date;
      rdfs:domain :Artwork ;
      rdfs:range xsd:dateTime .

:description rdf:type owl:DatatypeProperty ;
             rdfs:subPropertyOf dc:description;
             rdfs:domain :Artwork ;
             rdfs:range xsd:string .

:title rdf:type owl:DatatypeProperty ;
       rdfs:subPropertyOf dc:title;
       rdfs:domain :Artwork ;
       rdfs:range xsd:string .
```

1.7 *Benefits of applying Semantic Web Technologies*

To summarise, below we provide a short list of the benefits of using semantic web technologies. What can semantic web actually mean for your institution?

1. It allows resource-based work. This means that as an institution, you can focus on describing the resources that are core to your institution and to reuse information that is core to other institutions. It can but benefit the quality of the resource descriptions. For contemporary art institutions, this means that you can focus on describing your artworks, their digital representations (surrogates), documentation and even the artists.

2. RDF is very flexible and extensible. It allows for easy reuse of information on the Web. For instance, instead of describing artists in your institution's catalogue, you can refer to artist descriptions on the Web, reusing data from other institutions that are maybe more expert in describing that specific type of resource.

3. Because RDF is very flexible and extensible (and resource-based), using this technology enhances the contextualisation of your descriptions. All your resources can be linked to external resources on the Web, supporting the disambiguation of resource information.

4. Extra context information comes from your data model, which injects semantics into your RDF graph. This data model can be linked to other data models to promote the interoperability and is also a way of enhancing the semantics of your data model.

5. A lot of information is actually included in your data model. Because it's also extensible, extending your data model can extend your system, without even having

to do expensive updates to it. Thus, not only your data model is highly extensible, but your information system also becomes highly extensible with minimal effort.

6. Your data model allows for machines to interpret your data. This means machines (often reasoners) can derive a lot of extra information, e.g., classifications of your artworks, if your data model supports it.

2 *Contextualisation of Contemporary Art*

2.1 *Introduction*

In the previous chapter, we introduced some basic semantic web technologies. These are especially suited for the contextualisation of the description of things, including contemporary art. Three building blocks of the Semantic Web are key to contextualisation of resource descriptions:

- URIs:
  URIs identify a resource uniquely.
- RDF
  This framework represents data as a graph, of which the nodes, and even the edges, are URIs. This makes both the resources and the vocabularies describing them unique, allowing for semantics and disambiguation.
- RDFS / OWL:
  These models will actually make up your data model and will inject the RDF graph with semantics.

In this chapter, we will focus on how to use semantic web technologies to accommodate this contextualisation and the enrichment of cultural heritage. There are basically three ways of achieving a more contextualised cultural heritage description:

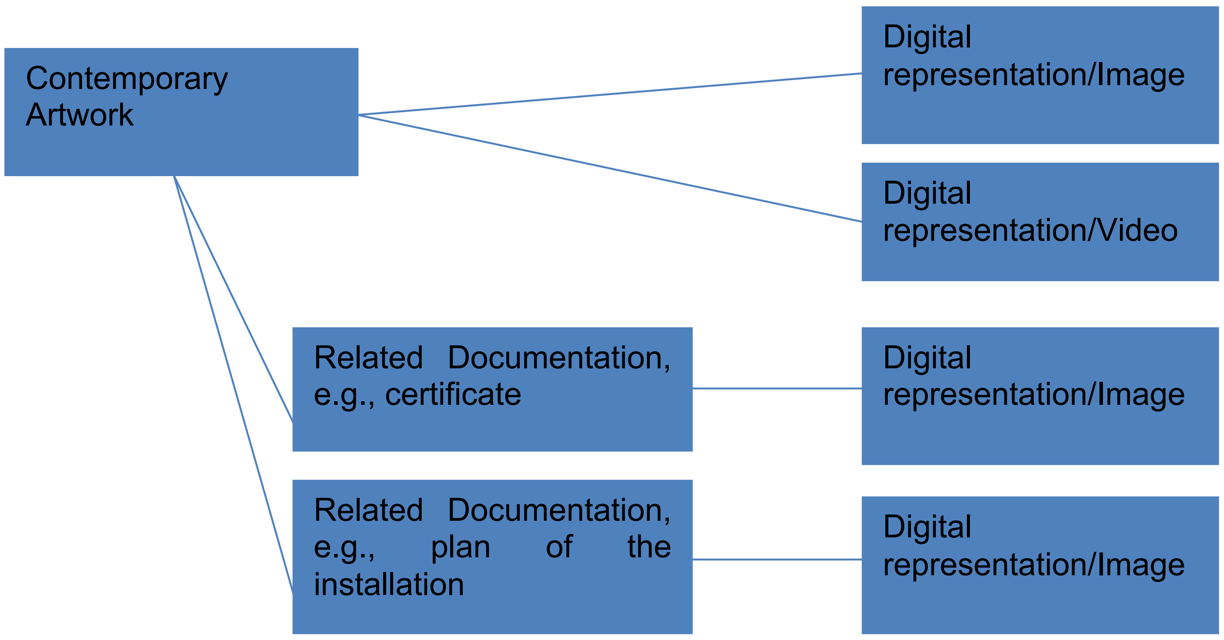
- data model / ontology;
- data descriptions;
- enrichment.

Each way of contextualising will be discussed in the following sections. As a starting point, the recommendations of D3.1 *Metadata implementation guidelines for digitised contemporary artworks* are used.

2.2 *Data Model*

When it comes to describing things, the Semantic Web provides a different perspective from traditional document based technologies. The Semantic Web and RDF is resource-based, not document-based. Every resource needs a URI and a description, of course.

In D3.1 the following schema gives an overview of a typical contemporary art object. Such an object consists of the contemporary artwork description, a digital representation of the artwork, such as an image and a video, related documentation of the artwork, such as a certificate and the digital representations of the related documentation.


Contemporary Artwork
Digital representation/Image
Digital representation/Video
Related Documentation, e.g., certificate
Digital representation/Image
Related Documentation, e.g., plan of the installation
Digital representation/Image

A distinction can already be made in these parts of a contemporary art object:

- contemporary artwork and related documentation;
- digital representations.

Both categories need different kinds of metadata. The contemporary artwork and related documentation need descriptive metadata. Their digital representations need technical metadata. And all the metadata needs to be related in a proper way.

For **contemporary art**, there are some **specific requirements**:

- Complex artworks:
  Contemporary artwork can be very complex. An installation can be an artwork, but every part of the installation can also be an artwork in itself.
- No distinction between a contemporary artwork and related documentation:
  Sometimes the related documentation of an artwork, e.g., a certificate, is the artwork itself. This might be the case when the artwork is a conceptual piece (that for instance results in a performance);
- Technical metadata:
  Contemporary art can rely on very specific hardware / software to be rendered. This needs to be described in the technical metadata. In certain cases, the look-and-feel also needs to be described. For instance, when a video artist has fixed a certain projection size and sound level for exhibiting his work, these features need to be described in order for them not to be lost in the future.

These requirements have to be kept in mind when designing the data model. There are two ways of defining the data model:

- reuse existing vocabularies / ontologies;
- design your own vocabulary / ontology.

.

***Best practice**: When defining your data model, reuse existing vocabularies / ontologies wherever possible.*

The best thing to do is to reuse existing vocabularies / ontologies. These vocabularies are made by domain specialists and incorporate a lot of the best practices of that domain. Not all vocabularies / ontologies will cover all aspects of a resource, especially not when it comes to administrative data. This is not a problem for RDF, because it is designed to be extensible. As such, every vocabulary / ontology that you want to use in your data model can be extended. When extending an existing model, add semantics for every extension. In practice, this means you should develop your own RDFS or OWL model, with its own namespace, to declare the semantics of your extensions.

***Best practice**: When extending an existing model, define a RDFS or OWL model with a namespace you control to declare the extensions.*

First, we will provide an introduction on how to reuse existing vocabularies, which ones to reuse, and how to tackle specific contemporary art requirements. Later we will explain how to model your own data model.

2.2.1 Reuse of vocabularies / ontologies

The first thing you have to do is to identify the resources you want to describe. We have already identified the fact that we need metadata for:

- artworks;
- related documentation;
- digital representations.

Additionally, extra resources can be identified depending on the information you have on the artworks. This is what is typically called contextual information.

- artists;
- locations;
- timespans;
- vocabularies;
- events;
- companies / institutions;
- …

Best practice: Identify the types of resources in your descriptions.

Each of these resources needs to have a URI and a description. Anyone can create their own URIs, as long as they follow the specification. These URIs will uniquely identify resources on the Web. A more strategic decision is the choice of data model.

When reusing existing vocabularies / ontologies, one can opt for a data model that already captures a lot of these resource descriptions. An example of this kind of data model is the Europeana Data Model (EDM). In EDM the following class instances are present:

Core EDM classes

• edm:ProvidedCHO	the provided cultural heritage object.
• edm:WebResource	the web resource that is a digital representation.
• edm:Aggregation	the aggregation that groups classes together, e.g., complex artworks.

Contextual EDM classes

• edm:Agent	the related agents (persons, organisations).
• edm:Place	the related locations.
• edm:Timespan	the related timespans.
• skos:Concept	the related SKOS (10) concepts.

As such, EDM has modelled every type of resource itself, except for controlled vocabulary terms, for which it has chosen the Simple Knowledge Organization System (SKOS).

Another possibility is the GAMA model[6], as explained in D3.1 *Metadata implementation guidelines for contemporary art*. This model was designed to describe media art, a specific category of contemporary art. The entities are described using the following classes:

• gama:Work	artworks, events and other data sources.
• gama:Person	persons, institutes, collectives.
• gama:Manifestation	physical representations of artworks
• gama:Archive	archives
• gama:Collection	collections of artworks

GAMA has also produced a GAMA vocabulary to annotate media art. The GAMA model uses this vocabulary to classify its media art and documentation.

Another solution would be to find an appropriate vocabulary / ontology for every type of resource. An extreme example would be to use Dublin Core to describe every type of resource and to interrelate these descriptions. Of course, there are more suitable models out there to be reused. Dublin Core is very generic. In D5.2 *Enrichment module and POC* some very useful vocabularies were listed that can easily be reused. This list repeated here, although slightly extended and organised differently.

The first thing we need is a description of **contemporary artworks** (and also **related documentation**). The following ontologies can be reused to describe these artworks:

- EDM (11):
A model that is targeted at describing cultural heritage information as LOD. It is developed by and for *Europeana* and is closely related to the LIDO schema.
- OpenART (12):
An event-driven ontology produced to describe 'art world' datasets. The ontology is split into a number of parts to allow greater re-usability. The ontology is linked to the Dolce Ultra Lite (DUL) ontology for greater applicability and interoperability.

[6] http://www.gama-gateway.eu/

- CIDOC-CRM (13):
The CIDOC Conceptual Reference Model (CRM) provides definitions and a formal structure to describe the concepts and relationships used in the documentation of cultural heritage assets.

In our data model example, we will start by reusing the edm:ProvidedCHO class. We actually define our own subclass ContemporaryArtwork, which are all the events previous to 1945. In EDM this is given by the Dublin Core property dc:date, which has preferably a range of edm:Timespan, e.g., 1945-01-01.

```
:ContemporaryArtwork rdf:type owl:Class ;
                     rdfs:subClassOf edm:ProvidedCHO,
                                    [ rdf:type owl:Restriction ;
                                      owl:onProperty dc:date ;
                                      owl:someValuesFrom [ rdf:type rdfs:Datatype ;
                                                           owl:onDatatype xsd:dateTime ;
                                                           owl:withRestrictions    (    [
xsd:minExclusive "1945-01-01"
                                                                                    ]
                                                                                  )
                                                         ]
                                    ] .
```

Because for contemporary art the **related documentation** can also be an artwork, we use edm:providedCHO to describe both. This way, we can define a separate class for related documents that can be distinguished from the contemporary artwork class.

```
:RelatedDocumentation rdf:type owl:Class ;
                     rdfs:subClassOf edm:ProvidedCHO .
```

To make a relation between the two resources (contemporary artwork and related documentation), we define a property relatedDocumentation. Its domain and range are ContemporaryArtwork. The property relatedDocumentation can be defined as follows:

```
:relatedDocumentation rdf:type owl:ObjectProperty ;
      rdfs:subPropertyOf rdfs:seeAlso ;
      rdfs:domain :ContemporaryArtwork ;
      rdfs:range  :RelatedDocumentation .
```

Sometimes, these artworks can be very **complex**. To describe such complex objects, one needs an ontology for describing aggregations or collections. A good candidate for this is:

- OAI-ORE: The OAI-ORE standard[7] has developed a standardised, interoperable and machine-readable mechanism that can express the information of compound objects, as discussed in deliverable 3.1 “Metadata Implementation Guidelines for Contemporary Art”.

The next thing we do is finding a model to describe the **artists** of the contemporary artwork. To describe persons, organisations and even software an effective ontology is:

[7] http://www.openarchives.org/ore/1.0/vocabulary.html

- Friend-Of-A-Friend (FOAF, (14)): a vocabulary for describing persons and organisations, e.g., the first and last name of a person, his/her social network accounts, contact information, etc.

We can define our own subclass of foaf:Person to describe artists, as shown in the following example:

```
:Artist rdf:type owl:Class ;
        rdfs:subClassOf foaf:Person ,
                                    [ rdf:type owl:Restriction ;
                          owl:onProperty dcterms:created ;
                          owl:onClass edm:ProvidedCHO ;
                          owl:minQualifiedCardinality "1"^^xsd:nonNegativeInteger
                        ] .
```

One step further is to define our own class of contemporary artists by “subclassing” the Artist class we just created.

```
:ContemporaryArtist rdf:type owl:Class ;
                    rdfs:subClassOf :Artist ,
                                    [ rdf:type owl:Restriction ;
                                      owl:onProperty dcterms:created ;
                                      owl:onClass :ContemporaryArtwork ;
                                      owl:minQualifiedCardinality
"1"^^xsd:nonNegativeInteger
                                    ] .
```

Now, we’ve composed the main parts of our descriptive metadata, but we still need to describe our **digital representations**. For these digital representations we need technical metadata. Good technical metadata models are:

- Media Annotations:
The Ontology for Media Resources 1.0 (15) is both a core vocabulary (a set of properties describing media resources) and its mapping to a set of metadata formats currently describing media resources published on the Web.
- PREMIS:
Preservation Metadata Implementation Strategies (16) is actually preservation metadata, with a semantic binding. This metadata has a high focus on technical metadata, i.e., premis:Object class. This vocabulary has the additional benefit of being able to describe the features to preserve the look-and-feel, i.e., significant properties, the hardware and software environments needed to render certain media art.

In our example we will make use of the premisowl:Object class to model digital representations. To integrate this into our data model, we can subclass the contemporary artwork class and the related documentation class to premisowl:IntellectualEntity. This way, we can use the property premisowl:linkingObject to link a contemporary artwork or related documentation to the premisowl:Object instance. This results in the following classes:

```
:ContemporaryArtwork rdf:type owl:Class ;
                     rdfs:subClassOf edm:ProvidedCHO,premisowl:IntellectualEntity,
                                     [ rdf:type owl:Restriction ;
                                       owl:onProperty dc:date ;
                                       owl:someValuesFrom [ rdf:type rdfs:Datatype ;
                                                            owl:onDatatype xsd:dateTime ;
                                                            owl:withRestrictions    (    [
xsd:minExclusive "1945-01-01"
```

```
                                                                                    ]
                                                                                  )
                                                              ]
                                          ] .


:RelatedDocumentation rdf:type owl:Class ;
                    rdfs:subClassOf edm:ProvidedCHO, premisowl:IntellectualEntity .
```

The next step is to introduce **vocabularies** for certain fields. A vocabulary can add some support for multilingual search in your system, as well as the classification of your resources, i.e., mainly artworks or related documentation.

Best practice*: When defining the properties, define their domains and ranges, as that will do the datatyping in the model. Restrict definitions tp free text fields when datatyping, as it is good for titles, names, descriptions, crowdsourced tags, and comments. Free text is hard to interpret for machines. A solution for this, simultaneously adding more context information, is the use of controlled vocabularies, e.g., SKOS vocabularies.*

A vocabulary is also a means of separating enumerations from your ontology. Such enumerations change often: their members can change as can the hierarchy of its members can change. This way, you only need to change your vocabulary and not your ontology / data model. A good model for the description of **controlled vocabularies** is:

- SKOS:
  SKOS allows one to formally describe the content and structure of structured vocabularies, such as thesauri, taxonomies, classification schemes, etc. Because SKOS is based on RDF (1), a Knowledge Organization System (KOS) that is defined using SKOS is machine-readable and publishable on the World Wide Web. SKOS is discussed in detail in Section 3.

Some good vocabularies to use for annotating contemporary art are for instance:

- the GAMA vocabulary:
  It is not modelled in SKOS, but as an enumerated class. The classes for classifying contemporary art are gama:WorkType and gama:MediaType.
- Wordnet:
  Wordnet[8] is a widely used lexical resource in natural language processing and information retrieval. It has also been adopted in the Semantic Web research community. It is used mainly for annotation and retrieval in different domains such as cultural heritage product catalogues and photo metadata.

As an example, our contemporary artworks will be linked to a SKOS vocabulary for classifying contemporary artworks. To do so, we will link on the DCA vocabulary, introduced in Section 4. It involves creating a subproperty of dc:subject and linking it to the DCA vocabulary. We'll also redefine our ContemporaryArtwork class so that it has at least one such property.

```
:contemporaryArtworkType rdf:type owl:ObjectProperty ;
      rdfs:subPropertyOf dc:subject;
```

[8] http://www.w3.org/TR/wordnet-rdf/

```
      rdfs:domain :ContemporaryArtwork ;
      rdfs:range [ rdf:type owl:Restriction ;
                   owl:onProperty skos:inScheme ;
                   owl:hasValue <http://purl.org/DCAVocabulary/>;
                 ] .


:ContemporaryArtwork rdf:type owl:Class ;
                     rdfs:subClassOf edm:ProvidedCHO, premisowl:IntellectualEntity
                               [ rdf:type owl:Restriction ;
                                 owl:onProperty dc:date ;
                                 owl:someValuesFrom [ rdf:type rdfs:Datatype ;
                                                      owl:onDatatype xsd:dateTime ;
                                                      owl:withRestrictions       (       [
xsd:minExclusive "1945-01-01"
                                                                                 ]
                                                                              )
                                                    ]
                               ],
                               [ rdf:type owl:Restriction ;
                                 owl:onProperty :contemporaryArtworkType ;
                                 owl:minQualifiedCardinality "1"^^xsd:nonNegativeInteger ;
                             ] .
```

Now we have defined a data model, which covers the most important types of resources for describing contemporary art. For extra context information, one can also detect the following types of resources in one's contemporary art descriptions.

For describing **locations** in the artworks' description, a good vocabulary to use is:

- Basic Geo (WGS84 lat/long) Vocabulary (17):
Basic GEO Vocabulary is a very basic vocabulary expressing the longitude and latitude of a location. It is very basic, but widely spread and powerful in combination with Dublin Core or FOAF.
- edm:Location :
This EDM class can also be reused for denoting places.

Other vocabularies that are good to reuse for describing **events** are:

- Linking Open Description of Events (LODE, (18)):
LODE is a basic model that can describe events. It describes the event itself, when it took or will take place, where, which actors were/are involved, etc.

And for **products**:

- GoodRelations (19):
GoodRelations is a model that describes products and services offered for e-commerce. It is capable of describing all the details of products, such as price, stock, etc.

Good **cross-domain** ontologies that can be used to describe anything are:

- Dublin Core (DC, (20)):
DC is a very generic model that can describe basic features (who, what, where, and when) about anything. The core model exists of fifteen properties.

- Dolce Ultra Lite (DUL, (21)):
  DUL is a lightweight upper ontology that describes general concepts across all knowledge domains. Such upper ontology is used often to unify data, described using various ontologies.

If we put all the pieces of our ontology together, we get the following data model. First, we must import the ontologies we reuse. In our example, these are EDM and PREMIS. Next, we define a class ContemporaryArtworks, as the subclass of edm:ProvidedCHO, to inherit its description. At the same time, we must also make it a subclass of premisowl:IntellectualEntity so that we can attach a premisowl:Object to the premisowl:IntellectualEntity using the premisowl:linkingObject property. This class will describe our technical metadata (including the software and hardware environment needed to render the digital representation, as well as the look-and-feel). Next our ContemporaryArtwork class needs to have at least one DCA vocabulary term to classify it, and its dc:creator property must refer to an instance of the ContemporaryArtist class.

```
@prefix : <http://example.org/voc/vocabulary#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix dc: <http://purl.org/dc/elements/1.1/> .
@prefix dcterms: <http://purl.org/dc/terms/> .
@prefix dcterms: <http://purl.org/dc/terms/> .
@prefix premisowl: <http://multimedialab.elis.ugent.be/users/samcoppe/ontologies/
                Premis/premis.owl#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix voc: <http://example.org/voc/vocabulary#> .
@prefix xml: <http://www.w3.org/XML/1998/namespace> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@base <http://example.org/voc/vocabulary> .

<http://example.org/voc/vocabulary> rdf:type owl:Ontology ;
                                    owl:imports <http://www.europeana.eu/schemas/edm/>;
                                    owl:imports <http://multimedialab.elis.ugent.be/users/
                                                samcoppe/ontologies/Premis/premis.owl#>.

:ContemporaryArtwork rdf:type owl:Class ;
                     rdfs:subClassOf edm:ProvidedCHO, premisowl:IntellectualEntity,
                              [ rdf:type owl:Restriction ;
                                owl:onProperty dc:date ;
                                owl:someValuesFrom [ rdf:type rdfs:Datatype ;
                                                     owl:onDatatype xsd:dateTime ;
                                                     owl:withRestrictions       (       [
xsd:minExclusive "1945-01-01"
                                                                             ]
                                                                           )
                                                   ]
                              ],
                              [ rdf:type owl:Restriction ;
                                owl:onProperty :contemporaryArtworkType ;
                                owl:minQualifiedCardinality "1"^^xsd:nonNegativeInteger ;
                              ] ,
                              [ rdf:type owl:Restriction ;
                                owl:onProperty dc:creator ;
                                rdfs:range :ContemporaryArtist ;

                              ] .

:RelatedDocumentation rdf:type owl:Class ;
                     rdfs:subClassOf edm:ProvidedCHO, premisowl:IntellectualEntity .
```

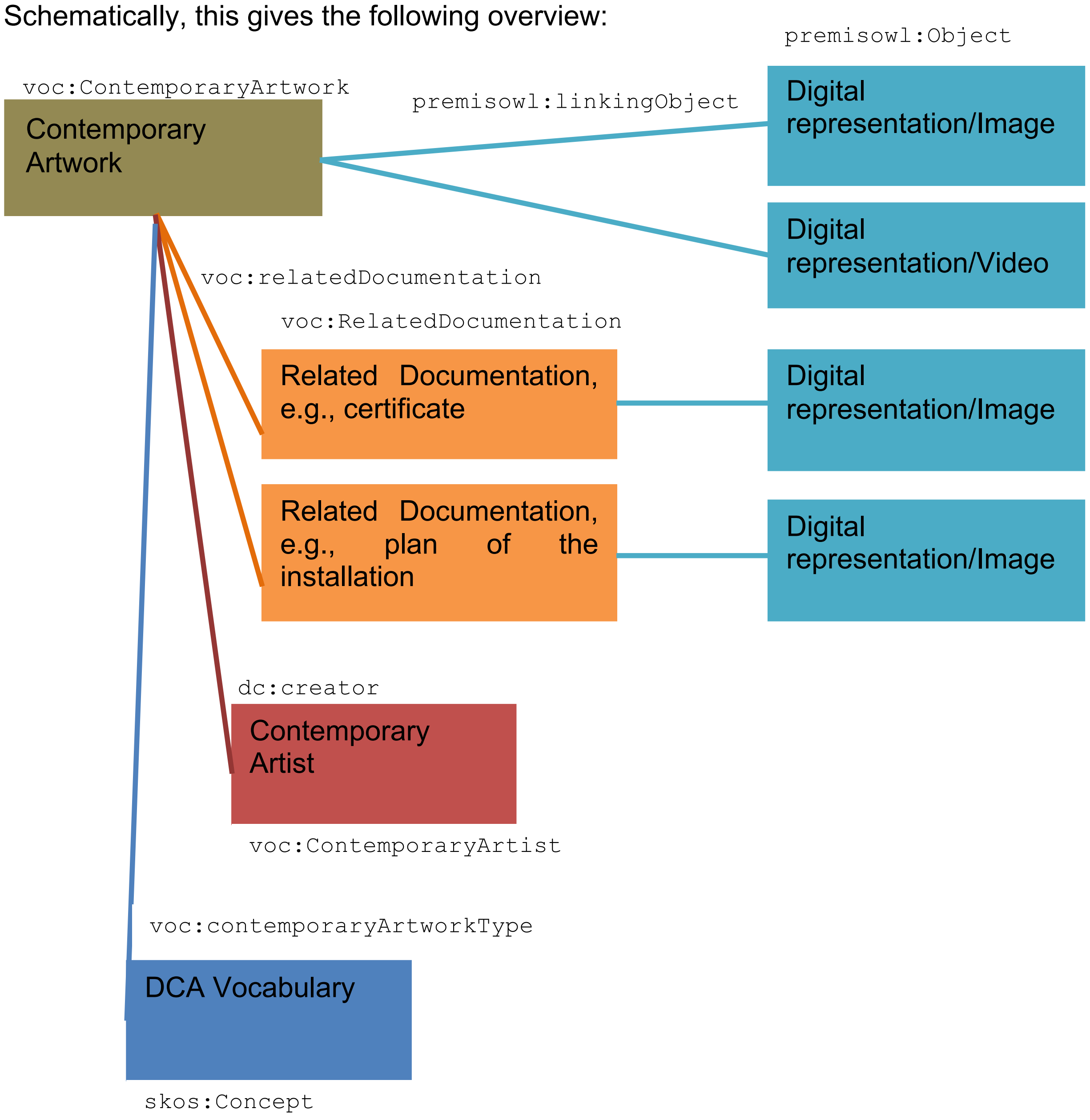
```
:Artist rdf:type owl:Class ;
        rdfs:subClassOf foaf:Person ,
                                        [ rdf:type owl:Restriction ;
                          owl:onProperty dcterms:created ;
                          owl:onClass edm:ProvidedCHO ;
                          owl:minQualifiedCardinality "1"^^xsd:nonNegativeInteger
                        ] .

:ContemporaryArtist rdf:type owl:Class ;
                    rdfs:subClassOf :Artist ,
                                    [ rdf:type owl:Restriction ;
                                      owl:onProperty dcterms:created ;
                                      owl:onClass :ContemporaryArtwork ;
                                      owl:minQualifiedCardinality
"1"^^xsd:nonNegativeInteger
                                    ] .

:contemporaryArtworkType rdf:type owl:ObjectProperty ;
      rdfs:subPropertyOf dc:subject;
      rdfs:domain :ContemporaryArtwork ;
      rdfs:range [ rdf:type owl:Restriction ;
                   owl:onProperty skos:inScheme ;
                   owl:hasValue <http://purl.org/DCAVocabulary/>;
                 ] .

:relatedDocumentation rdf:type owl:ObjectProperty ;
      rdfs:subPropertyOf rdfs:seeAlso;
      rdfs:domain :ContemporaryArtwork ;
      rdfs:range  :RelatedDocumentation.
```

Schematically, this gives the following overview:


premisowl:Object
voc:ContemporaryArtwork
premisowl:linkingObject
Contemporary Artwork
Digital representation/Image
Digital representation/Video
voc:relatedDocumentation
voc:RelatedDocumentation
Related Documentation, e.g., certificate
Digital representation/Image
Related Documentation, e.g., plan of the installation
Digital representation/Image
dc:creator
Contemporary Artist
voc:ContemporaryArtist
voc:contemporaryArtworkType
DCA Vocabulary
skos:Concept

2.2.2 Designing your own vocabulary / ontology

Another solution is to design your own data model. Some considerations need to be taken into account when going for this solution:

- RDFS or OWL knowledge is a prerequisite:
  Of course, when designing your vocabulary / ontology, you need to know about RDFS or OWL. RDFS is much easier, but not as expressive as OWL. OWL is more difficult, but more expressive. In the end, you will need to publish your data model, so that others can link on your model and other institutions can reuse it.
- Interoperability:
  When designing your own data model, you need to make sure it remains interoperable with existing vocabularies / ontologies.
- Maintaining your data model:

Data models evolve over time. This means you will need to maintain your data model. Certainly in the Semantic Web, where reuse is a key factor. Maybe some other institutions are reusing your data model. Your published data model will therefore need to be maintained.

First thing to do when designing your data model is to define the expressiveness your data model needs. This is determined by the reasoning your platform will need (in other words: the functionalities your data model needs to support). Based on this consideration, one can opt for:

- RDFS;
- OWL (EL, RL, QL).

Best practice*: Define your modelling language (RDFS, OWL) beforehand and try to stick to this language when designing your data model. For describing contemporary artworks, the best modelling language is OWL. RDFS is much lighter, but doesn't have enough expressivity. For instance, in RDFS one can't model an artwork has got to have at least one creator.*

The next step is to define first your classes. Mostly, the type of resources detected in your information will all become classes in your data model. Once you have defined your classes, you can model their properties. These properties will play a crucial role in disambiguating and enriching your resources and enriching your resources as will be explained in Section 3.3 'Data Descriptions'. Finally, define the relationships between the different classes.

Best practice*: First, define the classes of your data model. Then, for each class you can define its properties for describing the instance of this class. Finally, define the relationships between your classes.*

Now, you have designed your data model, but it is yet not interoperable. The next thing to do, is to define mappings between your data model and existing ones. Doing this will already give you a basic form of interoperability. At the same time, these mappings act as a form of context information on your data model.

Best practice*: Define mappings between your data model and existing ones to achieve some basic interoperability.*

At this point, your model is ready for publication. Publishing your data model is crucial in letting other institutions reuse your data model (or parts of it). At the same time, other data models can define mappings to yours, contributing to its interoperability.

Best practice*: Publish your data model with a namespace you control.*

Maintaining you data model after it is designed is also crucial. It is not static. The requirements for your resource descriptions can change, the models you have defined mappings for can change and new models you want to define a mapping for can also pop up. As a result, after publishing your data model, it needs to be maintained.

Best practice*: Maintain your published data model.*

2.3 *Data Descriptions*

Each type of resource will need to be described in an unambiguous way. This means that for each resource, you need to have enough information in order to unambiguously identify related resources on the Web during the enrichment phase. Your data model cannot always enforce this, because you will not always have the information to unambiguously describe the resource. However, it is something to strive for.

In D3.1 *Metadata implementation guidelines for digitised contemporary artworks* we gave an overview of identified types of resources and the properties for each type. These properties were ranked from minimum, recommended to additional. The tables below show the property lists for every sort of item, i.e., artworks, digital representations (surrogates), events, and actors. The values for fields denoted with an *asterisk* should be taken from a controlled vocabulary.

Artwork	Minimum	Recommended	Additional
ID	X		
Title	X		
Date		X	
Type	X		
Description		X	
Place		X	
Measurements			X
Collection		X	
Rights		X	
Language		X	
Subjects/keywords*	X		
Events			X
Surrogates			X

Dig. Representation	Minimum	Recommended	Additional
URL	X		
Description		X	
Language			X
Date		X	
Type*		X	
Rights		X	
Measurements			X
Format (mimetype)		X	

Event	Minimum	Recommended	Additional
Title	X		

Date		X	
Type*	X		
Description		X	
Place			X
Actor			X
Language		X	

Actor	Minimum	Recommended	Additional
Name	X		
Role*		X	
Place			X
Biography			X
Year of Birth		X	
Year of Death			X

Table 1: table of recommended fields to be offered by the contemporary art institutions

For every type of resource, in D5.2 *Enrichment module and POC* we specified a set of properties to identify the entity uniquely, i.e., identifying properties. These properties are essential for contextualising your resources. At a later stage, these properties will also be used to enrich the resources with external resources.

Locations
Identifying properties:

- name;
- geographical coordinates.

Persons/Institutions
Identifying properties:

- name;
- date of birth;
- place of birth.

Artwork type
Identifying properties:

- the literal itself.

Artwork
Identifying properties:

- artwork ID;
- data provider.

For each resource, we need to identify its identifying properties. These properties will become crucial in:

- describing the resource unambiguously;
- enriching the resource unambiguously.

***Best practice**: Define the identifying properties for each type of resource. When describing your resources, you should try to describe these properties as much as possible. If possible, enforce them through your data model.*

2.4 *Enrichment*

When contextualising descriptions, enrichment is the most powerful tool. Enrichment of your resources means that you will discover others on the Web that describe the same thing. They can then be linked to each other using the owl:sameAs property. Enrichments serve two purposes:

- Disambiguation:
  URIs are unique, but names of persons, locations, etc. are not. If your resource is linked via owl:sameAs to an external one it provides context information. For instance persons tend not to have unique names, like Damien Hirst. Besides the artist, there may be a number of people who are also called Damien Hirst. If your resource is linked to <http://dbpedia.org/resource/Damien_Hirst>, we know which Damien Hirst we are talking about.
- Linked Open Data Cloud:
  Just as hyperlinks act in the Web of documents, enrichments act in the Linked Open Data Cloud. They make connections to related information, turning the disparate graphs into one, i.e., the knowledge base of the Web, which could act in the future as a single database, as shown in Figure 4.

Three things are crucial for supporting enrichment:

- SPARQL:
  This is the query language and protocol for querying RDF data and, therefore link discovery.
- External data sources:
  These are external datasets you can query for discovering links.
- Identifying properties:
  These properties will make up your queries for the remote data sources for link discovery.

The following two subsections will elaborate more on SPARQL queries making use of the identifying properties and enrichments sources, i.e., external data source that can be used for enriching your data. This has already been discussed in D5.2 *Enrichment module and POC*, but is repeated here again for the sake of completeness.

2.4.1 SPARQL & identifying properties

In this section, we'll a look at the identifying properties of each type of resource and show an SPARQL query example that makes use of these identifying properties to discover links between resources.

Locations
Identifying properties:

- name;
- geographical coordinates.

```
PREFIX dbpediaprop: < http://live.dbpedia.org/property/>
PREFIX dbpedia: <http://dbpedia.org/resource/>
PREFIX dbpediaowl: <http://live.dbpedia.org/ontology/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT ?place
WHERE
{
?place a dbpediaowl:Place.
?place rdfs:label 'Rijeka'@en.
?place dbpediaowl:country dbpedia:Croatia.
}

Result: <http://dbpedia.org/resource/Rijeka>
```

Persons
Identifying properties:
- name;
- date of birth;
- place of birth.

```
PREFIX dbpediaprop: <http://live.dbpedia.org/property/>
PREFIX dbpedia: <http://dbpedia.org/resource/>
PREFIX dbpediaowl: <http://live.dbpedia.org/ontology/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?person
WHERE
{
?person a foaf:Person.
?person dbpediaprop:name 'Gerhard Richter@en.
?person dbpediaprop:dateOfBirth '1932-02-09'^^xsd:date.
?person dbpediaowl:birthPlace dbpedia:Dresden.
}

Result: <http://dbpedia.org/resource/Gerhard_Richter>
```

Artwork type
Identifying properties:
- the literal itself.

```
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
SELECT ?concept
WHERE
{
?concept a skos:Concept.
?concept skos:prefLabel 'Video Art'@en.
}
```

Artwork
Identifying properties:
- artwork ID;
- data provider.

```
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
SELECT ?artwork
WHERE
{
?artwork a ore:Aggregation.
?artwork edm:aggregatedCHO 'Hirst:1423'
?artwork edm:dataProvider 'MuZee'.
}
```

2.4.2 Enrichment Sources

Named entities

Named entities are just known concepts, like important events, places, persons, etc. Usually, named entities are extracted from free text.

OpenCalais[9]
OpenCalais is the most famous named entity extractor. You can send text to OpenCalais and it will filter all the recognised named entities from the text. OpenCalais can detect persons, locations, events, brands, music bands, etc. OpenCalais immediately links to DBpedia, Wikipedia, Freebase, Reuters.com, GeoNames, Shopping.com, IMDB, and Linked MDB. OpenCalais provides web services to annotate your content automatically with rich semantic metadata. Next to the web services, OpenCalais also publishes its content as LOD. OpenCalais can only deal with English, French and Spanish.

DBpedia Spotlight[10]
DBpedia Spotlight is a named entity extractor like OpenCalais. It will annotate your free text with links to DBpedia resources. One drawback is that it is currently only available in English.

Cross-domain

These cross-domain datasets have information on various sorts of things, e.g., persons, locations, movies, books, cultural heritage, etc.

DBPedia[11]
DBpedia publishes the knowledge extracted from Wikipedia as LOD. It makes its data available via an SPARQL endpoint. It has information on persons, places, creative works (music albums, films, video games, etc.), organisations, species and diseases. DBpedia provides localised versions of DBpedia in one hundred and eleven languages. DBpedia publishes its content as LOD and has a public SPARQL endpoint available for querying the dataset. DBpedia also provides RDF dumps, which can be downloaded and ingested into triple stores.

Freebase[12]
Freebase is very similar to DBpedia. It holds information on persons, locations, organisations, etc. The data from Freebase is collected from various data sources, such as

[9] http://www.opencalais.com/
[10] http://github.com/dbpedia-spotlight/dbpedia-spotlight
[11] http://dbpedia.org
[12] http://www.freebase.com/

ArXiv, CrunchBase, Eurostat, Wikipedia, IMDB, Library of Congress, etc. Freebase can be queries using MQL, their own developed query language. They also provide an REST API and also publish the content as LOD. RDF dumps of Freebase are also available.

OpenCyc[13]
OpenCyc is the open source version of CYC. CYC is a very large general knowledge base (including a reasoning engine). The CYC ontology contains 100,000s of terms and millions of assertions, relating the terms to each other, forming an (English) upper ontology, the domain of which is all of human consensus reality. OpenCyc provides an API for application development.

YAGO2[14]
YAGO2 is a large semantic knowledge base, derived from Wikipedia, WordNet, and GeoNames. Its knowledge covers ten million entities (persons, organisations, cities, etc.). YAGO2 provides dumps, but it can also be queried and browsed.

Vocabularies

WordNet[15]
WordNet is a large lexical database of English. It organises nouns, verbs, adjectives, and adverbs in sets of cognitive synonyms (synsets), each expressing a distinct concept. WordNet superficially resembles a controlled vocabulary. WordNet can be browsed and is also available as dump.

LCSH[16]
Library of Congress Subject Headings are a thesaurus of subject headings. These subject headings capture the essence of the topic of a document. It can be used as a controlled vocabulary to classify bibliographic records. LCSH are currently available as LOD, as a SKOS vocabulary and as a RDF document, but they do not provide a SPARQL endpoint. The RDF documents can be downloaded and stored in a local triple store to query the vocabularies using SPARQL.

Cultural Heritage

Europeana[17]
Europeana is a single access point to millions of books, paintings, films, museum objects and archival records that have been digitised throughout Europe. It is an authoritative source of information coming from European cultural and scientific institutions. The content from *Europeana* is available via an API. The LOD pilot from *Europeana* also makes a collection of the content available as LOD.

The Data Hub[18]
The Data Hub contains 4,293 datasets that you can browse, learn about and download. There are many datasets focusing on art in it. You can search for the dataset you need and

[13] http://www.opencyc.org/
[14] http://www.mpi-inf.mpg.de/yago-naga/yago/
[15] http://wordnet.princeton.edu/
[16] http://id.loc.gov/authorities/subjects.html
[17] http://www.europeana.eu
[18] http://thedatahub.org/

download the dataset to use it as an enrichment source. Examples of interesting datasets for the art sector are:

http://thedatahub.org/group/open-glam
http://thedatahub.org/group/art
http://thedatahub.org/dataset/freebase-visual-art
http://thedatahub.org/dataset/grants-for-the-arts-awards-arts-council-england
http://thedatahub.org/dataset/ukgac

Locations

GeoNames[19]
GeoNames is a geographical database, with information on all countries. It contains over eight million place names. The data is available as a dump, or via web services.

World Factbook[20]
The CIA's World Factbook provides information on the history, people, government, economy, geography, communications, transportation, military and transnational issues for 267 world entities. The World Factbook is available as dump to download.

Persons

VIAF[21]
Virtual International Authority File initiative is a joint project of several national libraries plus selected regional and trans-national library agencies. The project's goal is to lower the cost and increase the utility of library authority files by matching and linking widely-used ones and making their information available on the Web. This data source can be used to enrich persons, in particular artists.

Movies

Linked MDB[22]
The Linked Movie DataBase publishes LOD on movies. The Linked MDB also links to DBpedia, YAGO, Flickr wrappr, RDF Book Mashup, MusicBrainz, GeoNames, IMDB, Rotten Tomatoes and Freebase. The content is therefore published as LOD and an SPARQL endpoint is made available for querying the dataset.

Music

Music Brainz[23]
MusicBrainz is a music knowledge base. It contains information on artists, release groups, releases, recordings, works, and labels, as well as relationships between them. The data from MusicBrainz is free to download and is also offered in RDF for download.

BBC Music[24]

[19] http://www.geonames.org/
[20] https://www.cia.gov/library/publications/the-world-factbook/
[21] http://viaf.org/
[22] http://www.linkedmdb.org/
[23] http://musicbrainz.org/
[24] http://www.bbc.co.uk/music

BBC Music is also a music knowledge base. It contains information on all the artists ever played at the BBC or on one of their radio shows. It also gathers information from MusicBrainz and Wikipedia. The data is available via LOD, and web services.

Books

RDF Book Mashup[25]

The RDF Book Mashup makes information available about books, their authors, reviews, and online bookstores. The data is taken from data sources like Amazon, Google, and Yahoo. The information is published on the Web as LOD and the data can be queried using SPARQL.

[25] http://www4.wiwiss.fu-berlin.de/bizer/bookmashup/

3 SKOS

3.1 *Introduction*

In this chapter, we'll take a closer look at the Simple Knowledge Organization System (SKOS). SKOS is a data model defined by the Semantic Web Deployment Working Group, which is part of the Semantic Web Activity of the World Wide Web Consortium (W3C).

SKOS allows one to formally formally describe the content and structure of structured vocabularies, such as thesauri, taxonomies, classification schemes, etc. Because SKOS is based on the Resource Description Framework (RDF) (1), a Knowledge Organization System (KOS) that is defined using SKOS is machine-readable and publishable on the World Wide Web. This subsequently allows concepts to be linked with other concepts defined in the Web of Data and is important for creating a contextualised description. At the same time, SKOS is also a means to support multi-linguality. The terms of a SKOS are identified using URIs, but each term can have multiple labels in different languages to support multilingual search and classification.

SKOS can enhance descriptions dramatically. One of the best practices is to avoid free text as much as possible in descriptions. The reason for this is that it is hard to interpret for machine and, as such, is not good for supporting disambiguation. Consider tagging of artwork descriptions. When tags are defined as free text, machines will have hard time understanding the text and disambiguating the concepts mentioned in the text. For instance, Java can mean the island, the coffee or the programming language. It is ambiguous if encoded as free text. If you can give the island, the coffee and the programming language a different URI, with a different meaning behind it, disambiguation, even by machines, will be no problem. Using SKOS vocabularies can therefore greatly enhance the contextualisation of the descriptions.

Another benefit of SKOS is that it is very flexible and extensible. Typically, SKOS is used for controlled vocabularies. These tend to change often. SKOS allows for modelling them in such a way it can be backward compatible if the vocabulary is changed or extended. Terms can be added and even their hierarchy can be changed without affecting already existing descriptions making use of the vocabulary.

In this chapter, an overview is given of important aspects of SKOS that need to be taken into consideration when modelling a KOS in order to maintain consistency and follow all integrity conditions defined in the SKOS specification. Additionally, a number of best practices is formulated. In the following Chapter, these best practices are employed for designing a multilingual vocabulary for describing contemporary art supporting multilingual search in Europeana.

The SKOS specification is defined in (2). A primer introducing SKOS is available at (3). SKOS use cases are described in (4).

3.2 *SKOS vocabulary*

Table and Table give an overview of all terms introduced in the SKOS data model26. Table gives an overview of the classes introduced (names of classes start conventionally with a capital letter) while Table lists the properties (property names start conventionally with a lowercase character). The SKOS namespace URI is 'http://www.w3.org/2004/02/skos/core#'. Conventionally, the prefix used with this namespace is 'skos'.

URI	
skos:Concept	
skos:ConceptScheme	
skos:Collection	
skos:OrderedCollection	

Table **2: Classes introduced in SKOS specification**

URI	Domain	Range	Symmetric	Transitive
skos:inScheme	rdfs:Resource	skos:ConceptScheme		
skos:hasTopConcept	skos:ConceptScheme	skos:Concept		
skos:topConceptOf	skos:Concept	skos:ConceptScheme		
skos:altLabel	-	rdfs:Literal		
skos:hiddenLabel	-	rdfs:Literal		
skos:prefLabel	-	rdfs:Literal		
skos:notation	-	rdfs:Resource		
skos:changeNote	-	rdfs:Resource		
skos:definition	-	rdfs:Resource		
skos:editorialNote	-	rdfs:Resource		
skos:example	-	rdfs:Resource		
skos:historyNote	-	rdfs:Resource		
skos:note	-	rdfs:Resource		
skos:scopeNote	-	rdfs:Resource		
skos:broader	skos:Concept	skos:Concept		
skos:broaderTransitive	skos:Concept	skos:Concept		X
skos:narrower	skos:Concept	skos:Concept		
skos:narrowerTransitive	skos:Concept	skos:Concept		X
skos:related	skos:Concept	skos:Concept	X	
skos:semanticRelation	skos:Concept	skos:Concept		
skos:member	skos:Collection	union(skos:Concept, skos:Collection)		
skos:memberList	skos:OrderedCollection	rdf:List		
skos:broadMatch	skos:Concept	skos:Concept		
skos:closeMatch	skos:Concept	skos:Concept	X	
skos:exactMatch	skos:Concept	skos:Concept	X	X
skos:mappingRelation	skos:Concept	skos:Concept		
skos:narrowMatch	skos:Concept	skos:Concept		
skos:relatedMatch	skos:Concept	skos:Concept	X	

Table 3: Properties introduced in SKOS specification

The following sections give a detailed overview of the SKOS vocabulary.

[26] The SKOS data model is defined as an OWL Full ontology

3.3 *SKOS classes*

The SKOS specification defines four classes: Concept, ConceptScheme, Collection and OrderedCollection. Each class is defined as an instance of owl:Class (defined in the OWL specification). The following subsections describe each class in more detail.

3.3.1 The skos:Concept class

Instances of the skos:Concept class represent concepts that will be used in the KOS. The SKOS specification states that an instance of the Concept class can be viewed as an idea or notion; a unit of thought. This is a very flexible view. However, during KOS modelling it is not always that straightforward to decide what should be modelled as concept.

In order to introduce a SKOS concept, it is sufficient to state (in RDF) that a resource is an instance of the skos:Concept class, as illustrated below (Turtle notation).

```
<MediaArt> rdf:type skos:Concept .
```

3.3.2 The skos:ConceptScheme class

SKOS offers the means to represent a KOS using the skos:ConceptScheme class. A SKOS concept scheme can be viewed as an aggregation of one or more SKOS concepts. Semantic relationships between those concepts may also be viewed as part of a concept scheme. However (as will be discussed below), it can only be formally stated which concepts are part of a concept scheme. SKOS provides no means of stating which relationships between concepts are part of a concept scheme.

Instances of the skos:ConceptScheme class are used to aggregate those of the skos:Concept class. In order to introduce a SKOS concept scheme, it is sufficient to provide a resource with a type indication of skos:ConceptScheme, as illustrated below.

```
<ContemporaryArt> rdf:type skos:ConceptScheme .
```

In order to add a concept to a concept scheme, the skos:inScheme property is used.

```
<MediaArt> skos:inScheme <ContemporaryArt> .
```

The skos:inScheme property is defined as an instance of the class owl:ObjectProperty. The range (asserted by using the rdfs:range property defined in RDF Schema) of skos:inScheme is the class skos:ConceptScheme. Note that no domain is stated for the skos:inScheme property, i.e., the domain is the class of all resources (rdfs:Resource). This decision has been made to provide more flexibility.

It’s important to note that SKOS allows instances of the skos:Concept class to belong to more than one concept scheme. This is in contrast to many information systems, which do not allow using of concepts in more than one concept scheme. This flexibility allows new concept schemes to be defined by using concepts defined in other concept schemes.

```
ex:MediaArt        skos:inScheme           ex:ExternalConceptScheme .
ex:MediaArt        skos:inScheme           <ContemporaryArt> .
```

Integrity constraint: The extension (i.e., the set of instances of a class) of the skos:ConceptScheme class must be disjoint with the extension of the skos:Concept class. This means that a resource cannot simultaneously be an instance of the skos:Concept class and the skos:ConceptScheme class.

***Best practice**: As a concept scheme can contain a very large amount of concepts, a best practice is to formally state which concepts are positioned at the top in the hierarchy of the concept scheme.*

This can be done by using one of the following properties: skos:hasTopConcept (which is defined as a sub-property of skos:inScheme) or skos:topConceptOf. The properties skos:hasTopConcept and skos:topConceptOf are defined as inverse properties (using the owl:inverseOf property defined in OWL). This means that when the following fact is stated:

```
<ContemporaryArt> skos:hasTopConcept      <MediaArt> .
```

an OWL-reasoner will entail the fact:

```
<MediaArt>        skos:topConceptOf       <ContemporaryArt>  .
```

***Best practice**: In order to improve readability, it is advised to only use one of these two (skos:hosTopConcept or skos:topConceptOf) properties during modeling.*

Note that the use of the skos:hasTopConcept (or skos:topConceptOf) is a best practice and therefore there are no integrity conditions on their use. The consequence is that it is possible to define structures that do not seem logical, but are still consistent with the SKOS specification, as illustrated in the following example.

```
<ContemporaryArt> skos:hasTopConcept      <Animation>.
<Animation>       skos:broader            <MediaArt>.
<MediaArt>        skos:inScheme           <ContemporaryArt> .
```

Although this does not seem valid from a modelling perspective, this example is consistent with the SKOS specification.

For every concept that must be part of a concept scheme, a formal assertion is needed stating that the concept belongs to the concept scheme. For example, when a concept has been linked to a concept scheme (e.g., using the skos:inScheme property) and this concept has several narrower concepts, it does not imply that these narrower concepts are also part of the concept scheme, unless explicitly stated. This means for example that the following facts:

```
<MediaArt>          skos:narrower          <Animation>                           .
<MediaArt>          skos:inScheme          <ContemporaryArt> .
```

will not entail the fact

```
<Annimation>        skos:inScheme          <ContemporaryArt> .
```

Note that because skos:topConceptOf was defined as a subproperty of skos:inScheme, the usage of this property allows an RDFS reasoner to entail the fact that the concept belongs to the concept scheme.

```
<MediaArt>          skos:topConceptOf      <ContemporaryArt> .
```
entails
```
<MediaArt>          skos:inScheme          <ContemporaryArt> .
```

3.3.3 The skos:Collection class

An instance of the skos:Collection class can be used to group concepts. This can be useful to state that some concepts have something in common. This collection can then for instance be labelled or provided with additional information.

To introduce a collection, a resource has to be typed as being an instance of the class skos:Collection. Concepts can be added to this collection using the skos:member property.
```
<MyInterests> rdf:type skos:Collection ;
skos:member <Animation> , <VideoGameArt> , <MultimediaInstallation> .
```

The domain of the skos:member property is skos:Collection, the range is the union of the classes skos:Concept and skos:Collection. This allows nesting of collections.

3.3.4 The skos:OrderedCollection class

The skos:OrderedCollection class has a similar purpose as the skos:Collection class. However, it should be used when the order of the elements appearing in a collection is important.

The skos:OrderedCollection class is defined as a subclass of the skos:Collection class. Therefore, when a resource is typed as being an instance of the skos:OrderCollection class, it is understood that this resource is also an instance of the skos:Collection class.

An ordered collection is introduced by typing a resource with skos:OrderedCollection. Then a list is constructed containing all member concepts. This list is then linked to the instance of the skos:OrderedCollection class using the skos:memberList property. The domain of this property is the class skos:OrderedCollection, the range is rdf:List. A Collection instance can be inferred from a skos:OrderedCollection instance.

The skos:memberList property is defined as a functional property, as it is not logical that an ordered collection has multiple member lists. However, this is not an integrity constraint, as it’s not possible without explicitly stating that two lists are different.

3.3.5 Restrictions on class usage

Integrity constraint: The classes skos:Concept, skos:ConceptScheme and skos:Collection are disjoint. This means, for example, that it cannot be stated that a resource is an instance of either the skos:Concept or skos:ConceptScheme class once it has been stated that it is an instance of the skos:Collection class.

3.4 *SKOS properties*

3.4.1 Labelling properties

The SKOS specification defines three properties that can be used to provide resources with labels:

- skos:prefLabel;
- skos:altLabel;
- skos:hiddenLabel.

These properties are defined as instances of the owl:AnnotationProperty class defined in OWL (11) and as sub-properties of the rdfs:label property, which is defined in the RDFS (RDF Schema) vocabulary (8). As a result, the range of these properties is the rdfs:Literal class.

No domain has been specified for these properties. Therefore, they can be used to provide labels to any resource.

The property skos:prefLabel is used to denote a preferred lexical label for a resource. In an information system, this label will typically be presented to the user.

```
<MediaArt> skos:prefLabel "Media Art"@en.
```

Integrity constraint: A resource is not allowed to have more than one preferred label for a specific language tag.

The property skos:altLabel is used to provide alternative labels for a concept. These alternative labels represent valid alternative representations for a resource, but are not the preferred representation. Therefore, these may also be displayed to the user.

```
<MediaArt> skos:altLabel "MultimediaArt"@en.
```

Note that it is valid to provide multiple alternative labels for a specific language.

The property skos:hiddenLabel is used when a label is not expected to be shown to a user, but is to be used for example for text indexing. In this way, a misspelled label (or a label that has been deprecated as a consequence of spelling rule changes) can be provided to a resource allowing a search application to find the relevant resource.

Integrity constraint: The properties skos:prefLabel, skos:altLabel, skos:hiddenLabel are pairwise disjoint.

Best practice*: Every concept should have a preferred label.*

Best practice*: Following common practice in KOS design, the preferred label of a concept may also be used to represent this concept unambiguously within a KOS and its applications. So even though the SKOS data model does not formally enforce it, it is recommended that no two concepts in the same KOS have the same preferred lexical label for any given language tag.*

The object of triples using a labelling property is a plain literal. As mentioned in Section **Error! Reference source not found.**, a plain literal has an optional language tag. The language tag then indicates the language used in the literal (21). This indication is important for many types of information processing, which require knowledge of the language in which information is expressed in order for the information processing to be performed, e.g., spell-checking and computer-synthesized speech.

Best practice*: Always provide label tags to literals when using SKOS labelling properties.*

As it is best practice to always provide a language tag to labelling properties, this can lead to multiple modelling options. Consider the following example in which we wanted to label a concept with different labels in different languages.

1. **Minimal labelling approach**

```
<MediaArt> rdf:type skos:Concept;
      skos:prefLabel "Media Art"@en;
      skos:altLabel "Multimedia Art"@en;
      skos:altLabel "MM Art"@en;
      skos:prefLabel "Media Kunst"@nl;
      skos:altLabel "Multimedia Kunst"@nl;
      skos:altLabel "MM Kunst"@nl;
      skos:narrower <Animation>;
      skos:narrower <MultimediaInstallation>.

<Animation> rdf:type skos:Concept;
      skos:prefLabel "Animatie"@nl;
      skos:prefLabel "Animation"@en.

<MultimediaInstallation> rdf:type skos:Concept;
      skos:prefLabel "Multimedia Installatie"@nl;
      skos:prefLabel "Multimedia Installation"@en;
```

In this example, we provided both a Dutch and English preferred label to the concepts. In addition, we added abbreviations for these labels as alternatives. The decision to use the full label instead of the abbreviation as preferred label was taken because it is best practice to have unique preferred labels for each concept in the KOS (and using the full representation minimises the possibility of duplicate preferred labels). However, if the abbreviations are actually the preferred labels (this depends on the use scenario of the KOS), and no other concepts appear in the KOS with the same abbreviations, one can decide to use the abbreviations as preferred labels.

The model above does not state that the "Multimedia Art" label is also used in Dutch to refer to this concept. This is because, according to (21), the language tag indicates the language used in the label, which in this case is English.

When using this modelling approach, it is up to the application using the KOS to use labels from other languages in the absence of a label for a specific language (e.g., Dutch). When a user, for example, has selected Dutch as the preferred language to be used in the user interface, the application cannot present Dutch labels for the concepts above, as these are not present. One solution is to present English labels by default in the absence of a Dutch label.

2. **Maximal labelling approach.**

When it is not expected that the KOS will be subject to information processing applications (e.g., when the KOS is only used as a navigational assistance tool in a search application) and/or the application using the KOS is not able to select labels defined in other languages, it could be decided to explicitly model the labels that are used in Dutch, although the label itself is actually an English one.

```
<MediaArt> rdf:type skos:Concept;
      skos:prefLabel "Media Art"@en;
      skos:altLabel "Multimedia Art"@en;
      skos:altLabel "MM Art"@en;
      skos:prefLabel "Media Kunst"@nl;
      skos:altLabel "Multimedia Kunst"@nl;
      skos:altLabel "MM Kunst"@nl;
      skos:altLabel "Media Art"@nl;
      skos:narrower <Animation>;
      skos:narrower <MultimediaInstallation>.

<Animation> rdf:type skos:Concept;
      skos:prefLabel "Animatie"@nl;
      skos:altLabel "Animation"@nl.
      skos:prefLabel "Animation"@en.

<MultimediaInstallation> rdf:type skos:Concept;
      skos:prefLabel "Multimedia Installatie"@nl;
      skos:altLabel "Multimedia Installation"@nl.
      skos:prefLabel "Multimedia Installation"@en.
```

The choice between these two modelling approaches depends on the KOS usage.

3.4.2 Notation properties

A notation is a lexical code used to identify the concept of a concept scheme uniquely. Although in the Web of Data (and SKOS) the URI is the preferred way to identify a concept, notations can be useful as a bridging mechanism.

A notation can be added to a skos:Concept instance using the skos:notation property.

```
<Fiction> skos:notation "1.2.32"^^<GenreNotationDatatype>.
```

The skos:notation property is defined as an instance of the owl:DatatypeProperty class defined in OWL.

***Best practice**: Use only typed literals when providing notations. The (user-defined) datatype URI then denotes a system of notations or classification codes. When providing preferred labels for a concept, it is best practice to not use typed literals but plain literals.*

***Best practice**: No two concepts in a concept scheme should be given an identical notation.*

3.4.3 Properties denoting an hierarchical relation

The following properties are defined in SKOS to denote an hierarchical relation (i.e., to state that one concept is more general than another one):

- skos:broader
- skos:narrower
- skos:broaderTransitive
- skos:narrowerTransitive

These properties are all defined as instances of the owl:ObjectProperty class. skos:broader and skos:narrower are defined as inverse properties of each other. Similarly, skos:broaderTransitive and skos:narrowerTransitive are defined as inverse properties of each other.

The properties skos:broaderTransitive, skos:narrowerTransitive and skos:related are defined as sub-properties of the skos:semanticRelation property. The domain and range of the skos:semanticRelation property is skos:Concept. skos:broader is defined as a subproperty of skos:broaderTransitive. Similarly, the skos:narrower property is defined as a subproperty of skos:narrowerTransitive.

***Best practice**: Only the properties skos:broader and/or skos:narrower should be used during modelling. The properties skos:broaderTransitive and skos:narrowerTransitive should be used by an application to access the transitive closure of an hierarchy expressed with skos:broader and skos:narrower properties.*

***Best practice**: Only use the skos:broader and skos:narrower properties to assert a direct hierarchical link between two concepts.*

In this way, an application is able to retrieve the direct broader and narrower terms of a concept. This is why the properties skos:broader and skos:narrower are not defined as transitive. Consider the following example:

```
<MediaArt>              skos:narrower               <Animation> .
<Animation>             skos:narrower               <StopMotion> .
```

As the skos:narrower property was not defined as a transitive property, the fact

```
<MediaArt>              skos:narrower               <StopMotion> .
```

will not be entailed by a reasoner. If this was the case (i.e., when skos:narrower was defined as a transitive property), it would become impossible to retrieve only the direct narrower concepts.

However, in some applications, it can be important to retrieve all narrower (or broader) concepts, even if they are not directly related. For this reason, SKOS defined the skos:broaderTransitive and skos:narrowerTransitive properties. As the property names suggest, these properties are defined as transitive (by defining these properties as instances of the owl:transitiveProperty class). The properties skos:broader and skos:narrower are defined as a subproperty of the skos:broaderTransitive and skos:narrowerTransitive property respectively. As a result, after applying an OWL reasoner, the following facts (among others) will be required:

```
<MediaArt>            skos:narrowerTransitive         <Animation> .
<Animation>           skos:narrowerTransitive         <StopMotion> .

<MediaArt>            skos:narrowerTransitive         <StopMotion> .
```

The first two are the result of the fact that skos:narrower is defined as a subproperty of skos:narrowerTransitive. The second is entailed because the skos:narrowerTransitive property is defined as a transitive property.

Note that SKOS allows for multiple skos:broader and skos:narrower relations between concepts. For example, it is valid to assert the following:

```
<MediaArt>            skos:narrower                   <Animation> .
<Animation>           skos:narrower                   <StopMotion> .
<MediaArt>            skos:narrower                   <StopMotion> .
```

However, the above example is not considered best practice, as skos:narrower should only be used to link direct narrower concept.

***Best practice**: In order to improve readability, it is only advisable to use either the skos:broader or skos:narrower property during the modelling of a KOS. Choosing which property to use during modelling depends on the approach. If the modelling approach follows a top-down pattern, then using the skos:narrower property is preferred. Alternatively, when the modelling approach is bottom-up, using the property skos:broader is preferred.*

3.4.4 Properties denoting an associative relation

The property skos:related can be used to state that two concepts are related to each other, but neither is more general or specific (otherwise, the skos:broader or the skos:narrower property should be used). skos:related is defined as a symmetric property, but not as a transitive property.

Integrity constraint: skos:related is disjoint with skos:broaderTransitive (and consequently with skos:narrowerTransitive, skos:broader and skos:narrower). Although this integrity constraint is not formally stated in the SKOS ontology, when it’s not followed, the model is considered not consistent.

For example, the following is considered inconsistent, as there is a conflict between associative links and hierarchical links.

```
<MediaArt>                  skos:narrower                <Animation> .
<Animation>                 skos:narrower                <StopMotion> .
<MediaArt>                  skos:related                 <StopMotion> .
```

This is an important constraint that should be taken into consideration when developing a KOS using SKOS. If two resources are related using the property skos:related, there must not be a chain using skos:broader, skos:narrower (and also skos:broaderTransitive and skos:narrowerTransitive, but these last two should not be used during modelling anyway) between the two concepts.

Finally, note that the domain and range of the skos:semanticRelation property is defined as a skos:Concept. Therefore, this property (and properties defined as subproperties of skos:semanticRelation) cannot be used to document, for example, a skos:Collection, as this would render the model inconsistent.

3.4.5 Documentation properties

It is often desirable to provide resources with additional information. For this purpose the SKOS specification defines the following properties:

- skos:note
- skos:changeNote
- skos:definition
- skos:editorialNote
- skos:example
- skos:historyNote
- skos:scopeNote

skos:changeNote, skos:definition, skos:editorialNote, skos:example, skos:historyNote and skos:scopeNote are defined as sub-properties of the more general skos:note property. All properties are also defined as instances of the owl:AnnotationProperty class. As no domain is defined for these properties, they can be used to annotate every resource (a skos:Concept, a skos:ConceptScheme, a resource of type owl:Class, etc.). Also, no range is defined for these properties, allowing both RDF plain literals and other resources to appear as the object of the assertion. For example:

```
<ExampleConcept> skos:note "note containing additional information"@en .
<ExampleConceptSchem> skos:note <ConceptSchemeNoteResource>.
```

***Best practice**: The skos:note property is one that can be used for general documentation purposes. However, it is advised to use the more specialised properties when applicable.*

The following list explains when to use each property.

- Skos:scopeNote:
This is used to provide additional information related to the intended meaning/usage of the resource. This additional information may not be complete.
- skos:definition:
This is used to give a complete description of the intended meaning of the resource.
- skos:example:
This is used to give an example of the usage of the concept.
- skos:historyNote:
This is used to give information about the fact that the meaning of the resource has changed.
- skos:editorialNote:
This is used to provide general information for the editor of the KOS.

- skos:changeNote:
This is used to provide detailed information about modifications to the KOS.

The skos:editorialNote and skos:changeNote properties are intended to be used to provide information to the KOS editor(s) and not to their users.

3.4.6 Mapping properties

Mapping concepts defined in different concept schemes can be very useful in many applications. For this reason, the SKOS specification has introduced the following mapping properties:

- skos:mappingRelation;
- skos:exactMatch;
- skos:closeMatch;
- skos:broadMatch;
- skos:narrowMatch;
- skos:relatedMatch.

All properties stated above are defined as instances of the owl:ObjectProperty class. skos:closeMatch, skos:broadMatch, skos:narrowMatch and skos:relatedMatch are subproperties of the skos:mappingRelation. skos:mappingRelation is defined as a subproperty of skos:semanticRelation. Therefore, the domain and range of these properties is skos:Concept.

***Best practice**: When two concepts from different concept schemes denote the same concept, and therefore could be used interchangeably, the skos:exactMatch property can be used to map them. Note that the skos:exactMatch property is defined as a transitive property, and therefore care should be taken when using it in modelling. It should only be used when two concepts have a similar intended meaning over a broad range of applications and concept schemes.*

If this is not the case, i.e., the concepts can only be used interchangeably in some applications, the property skos:closeMatch should be used instead. The reason why skos:closeMatch is preferred in this case is that it is not defined as a transitive property. skos:exactMacth is defined as a subproperty of skos:closeMatch.

The skos:broadMatch and skos:narrowMatch can be used to define an hierarchical mapping between concepts defined in separate concept schemes. These properties are defined as inverse properties of each other. The skos:broadMatch and skos:narrowMatch are defined as subproperties of skos:broader and skos:narrower respectively.

Associative mapping between concepts defined in separate concept schemes is established using the skos:relatedMatch property. skos:relatedMatch is defined as a subproperty of skos:related.

The skos:closeMatch, skos:exactMatch and skos:relatedMatch are defined as symmetric properties.

Integrity constraint: skos:exactMatch is disjoint with the properties skos:broadMatch, skos:narrowMatch and skos:relatedMatch. A KOS is not considered consistent according to the SKOS data model when a clash can be found between exact mappings and associative or hierarchical mappings. The following example is therefore considered not consistent according to the SKOS data model.

```
@prefix my: <http://example.org/myvocab/> .
@prefix ext: <http://example.com/vocab/> .

<my:MediaArt> skos:exactMatch <ext:MediaArt> .
<my:MediaArt> skos:relatedMatch <ext:ContemporaryArt> .
```

OWL also provides some properties that can be used to map resources: owl:equivalentClass, owl:equivalentProperty and owl:sameAs. The owl:equivalentClass is used to assert that two classes have the same class extension. Similarly, the owl:equivalent property is used to assert that two properties have the same property extension. The owl:sameAs property is used to assert that two resources actually refer to the same thing. At first glance, it can seem useful to use the owl:sameAs property to map concepts from different concept schemes when they denote the same thing.

***Best practice**: However, because of the formal consequences using this property has, it is not recommended to use the owl:sameAs property to map SKOS concepts.*

Consider the following example as an illustration of the unwanted results using owl:sameAs can have.

```
@prefix my: <http://example.org/myvocab/> .
@prefix ext: <http://example.com/vocab/> .

my:MediaArt skos:prefLabel "Media Art"@en.
ext:Media-Art skos:prefLabel "Multimedia Art"@en.

my:MediaArt owl:sameAs ext:Media-Art.
```

After applying an OWL reasoner, the following additional fact (amongst others) will be required.

```
my:MediaArt skos:prefLabel "Multimedia Art"@en.
ext:Media-Art skos:prefLabel "Media Art"@en.
```

Note that now both resources have two different preferred English labels. As already noted, an SKOS model is considered to be inconsistent when a concept has more than one preferred label for a given language tag. This example illustrates why the use of skos:exactMatch is preferred over that of owl:sameAs.

***Best practice**: Use the mapping properties skos:broadMatch, skos:narrowMatch and skos:relatedMatch to define hierarchical and associative relations between concepts defined in different concept schemes and use the skos:broader, skos:narrower and skos:related properties to define hierarchical and associative relations between concepts defined in the same concept scheme.*

3.5 *SKOS design strategies*

3.5.1 Concept grouping

During modelling a KOS, it is often necessary to group concepts that have something in common. Below we explain two possible approaches to do this.

1. **Grouping concepts using collections.**

Consider the following example. The concept ‘animation’ has been introduced as the top-level concept of a concept scheme about contemporary art. Additionally, concepts such as ‘cartoon’, ‘stopmotion’, ‘animatronics’, etc. are introduced as narrower concepts of the ‘animation’ concept. Note that these concepts can be grouped. For example, the concepts ‘cartoon’ and ‘animatronics’ rely on computers for generating the animation, ‘stopmotion’ is hand-generated animation. This can be done using collections as illustrated below.

```
<Animation> rdf:type skos:Concept .
<Animation> skos:inScheme <ContemporaryArt> .
<Animation> skos:topConceptOf <ContemporaryArt> .

<cartoon> rdf:type skos:Concept .
<stopmotion> rdf:type skos:Concept .
<animatronics> rdf:type skos:Concept .

<cartoon> skos:inScheme <ContemporaryArt> .
<stopmotion> skos:inScheme <ContemporaryArt>.
<animatronics> skos:inScheme <ContemporaryArt>.

<Animation> skos:narrower <cartoon> .
<Animation> skos:narrower <stopmotion> .
<Anitmation> skos:narrower <animatronics> .

<HandGeneratedArt> rdf:type skos:Collection ;
skos:member <stopmotion> , <cartoon> ;
skos:prefLabel "Hand generated art."@en .

<ComputerGeneratedArt> rdf:type skos:Collection ;
skos:member <cartoon> , <animatronics> ;
skos:prefLabel "Computer generated art"@en .
```

As collections and concepts must be disjoint in SKOS, it is not possible to define semantic relations for a collection (because the range and domain of skos:semanticRelation is defined as skos:Concept.). Therefore, grouping concepts into collections has no influence on the position of a concept in the concept scheme. SKOS collections can be used to model hierarchies in which the notion indicated by a collection is not considered a real concept of the KOS.

It is up to the application using the SKOS model to decide how the concept hierarchy is displayed (e.g., whether or not to include collection membership information in the displayed hierarchy).

2. **Grouping concepts using hierarchical relations.**

SKOS concept schemes are often designed to be used as a navigation hierarchy (e.g., to assist users in a search application). If this is the case, it can be decided not to use collections but organise concepts using hierarchical relations only. This approach is illustrated in the following example.

```
<Animation> rdf:type skos:Concept .
<Animation> skos:inScheme <ContemporaryArt> .
<Animation> skos:topConceptOf <ContemporaryArt> .

<HandGeneratedArt> rdf:type skos:Concept ;
skos:prefLabel "Hand generated art."@en .

<ComputerGeneratedArt> rdf:type skos:Concept ;
skos:prefLabel "Computer generated art"@en .

<cartoon> rdf:type skos:Concept .
<stopmotion> rdf:type skos:Concept .
<animatronics> rdf:type skos:Concept .

<HandGeneratedArt> skos:inScheme <ContemporaryArt> .
<ComputerGeneratedArt> skos:inScheme <ContemporaryArt> .
<cartoon> skos:inScheme <ContemporaryArt> .
<stopmotion> skos:inScheme <ContemporaryArt> .
<animatronics> skos:inScheme <ContemporaryArt> .

<Animation> skos:narrower <ComputerGeneratedArt>.
<Animation> skos:narrower <HandGeneratedArt>.

<HandGeneratedArt> skos:narrower <stopmotion> , <cartoon> .
<ComputerGeneratedArt> skos:narrower <cartoon> , <animatronics> .
```

The decision as to which approach is followed is left to the KOS designer. This decision can be influenced by the intended use of the KOS (e.g., when the KOS will only be used as a navigation aid) and the application utilising the KOS (e.g., when an application does not support collection processing). However, from a pure modelling perspective, the first approach using collections is often considered better practice than the sole use of hierarchical relations, as it better models the semantics of the KOS.

Best practice*: nesting of concept schemes is not considered best practice. A concept scheme should be considered as an aggregation of concepts only. Therefore, if a concept is part of more than one concept scheme, it should be denoted through using multiple skos:inScheme properties: one for each concept scheme a concept belongs to.*

3.5.2 Poly-hierarchical knowledge organisation systems.

If we look at our previous example, 'cartoon' can be both computer-generated and hand-generated.

There are now two paths from the concept 'Animation' to the concept 'Cartoon': a path via 'HandGeneratedArt' and one via 'ComputerGeneratedArt'. This is valid modelling in SKOS and often used in poly-hierarchical knowledge organisation systems.

3.5.3 Modelling a KOS that changes over time

Adding concepts

OWL makes use of an *open world* assumption. As the SKOS data model has been designed as OWL ontology, this concept also applies to the SKOS data model. The open world assumption states that the truth-value of a statement is independent of whether or not it is known by any single observer or agent to be true.

For a SKOS concept scheme, this assumption means that a concept scheme only describes the concept scheme, but does not define it. In other words, it is possible that other concepts belong to the scheme, but are not yet known. Therefore, it is possible to add new concepts to a concept scheme along the way.

Removing concepts

When a concept no longer needs to be used for annotation purposes, it should also no longer be selectable in an annotation tool. However, it should still be discoverable in a search application, as items could have been annotated with this concept. For this reason, one approach could be to design two concept schemes; one containing all currently allowed concepts for annotation and one that contains all concepts that have been used in the earlier annotation process. In this way, concepts that are no longer in use are no longer available for annotation. However, during search, it is still possible to use these concepts.

Changing relations between concepts

A SKOS model is rather static, i.e., the SKOS data model only allows describing a KOS and does not provide a way to keep track of changes to the structure of the KOS, other than the use of notes.

Altering a SKOS model through changing relationships between concepts can have a major influence on the resulting set of information retrieval requests, as relationships between concepts are often used in order to construct a result set. Suppose for example that a concept ‘MediaArt’ has a narrower concept ‘Cartoon’. During annotation of a media item, only the concept ‘MediaArt’ was provided. When a user performs a search operation, which involves concept ‘MediaArt’, the item that was annotated with concept ‘Cartoon’ can also be relevant and therefore included in the result set through query expansion (e.g., including all narrower concepts of concept ‘MediaArt’ in the query). However, once this relation has been removed, it is no longer possible to retrieve the item using a query that involves concept ‘MediaArt’.

One way to alleviate this problem is to store the concept position in the model during annotation (i.e., its broader terms, narrower terms, transitive closure, etc.) as part of the annotation. The search application then uses this information instead of the current SKOS model. Changing the concept position in the SKOS model now no longer has an influence on the previously annotated media items, as the search application only uses the stored concept position instead of the current SKOS model. New annotations store the concept position in the current SKOS model.

KOS development approaches

Two different approaches can be used when introducing a KOS into a platform:

- Top down:
  This approach is used when the hierarchy of the KOS is known, but the complete set of concepts belonging to the KOS is not. Therefore, the hierarchy is first modelled and an initial set of concepts that are known to be part of the KOS is introduced. Additional concepts are then gradually added (e.g., to improve search operations).
- Bottom up:
  In this approach all concepts of the KOS are known, but their hierarchy is unknown. Additional relations are gradually introduced in order to further develop the KOS.

In practice, however, often a mix of the above-mentioned approaches is encountered. In this case, a KOS initially consists of a number of concepts and relations and additional ones are then gradually added to the KOS.

4 DCA SKOS Vocabulary

4.1 *Purpose*

In this chapter we're going to introduce the DCA vocabulary. It's designed using SKOS. Controlled vocabularies are recommended for some fields to contribute to the context of your descriptions. This holds true for artwork types, event types, surrogate types, keywords, and the roles of an actor. For dissemination to Europeana, we will provide a controlled vocabulary of keywords to denote the type of artworks as well as their related documentation.

This vocabulary is meant to support multilingual search in Europeana. Every partner institution describes the data in their language. This holds true also for their dissemination through Europeana. To create a common ground to search the whole DCA content, the DCA vocabulary was designed. It is not quite intended as a classification scheme for contemporary art. For this purpose, the vocabulary is too concise. In time, it could evolve to a classification scheme, as the vocabulary gets extended.

4.2 *Terms*

To design the vocabulary, we take a bottom up approach. First we collect all the terms that should be included in the vocabulary and then start to model their hierarchy. This set of terms is achieved through analysing the terms the content partners used to classify their artworks.

All used keywords are displayed in the tag cloud below. The bigger the word, the more it is used by different contemporary art institutions. The words that are used by at least two institutions are shown in the table below.

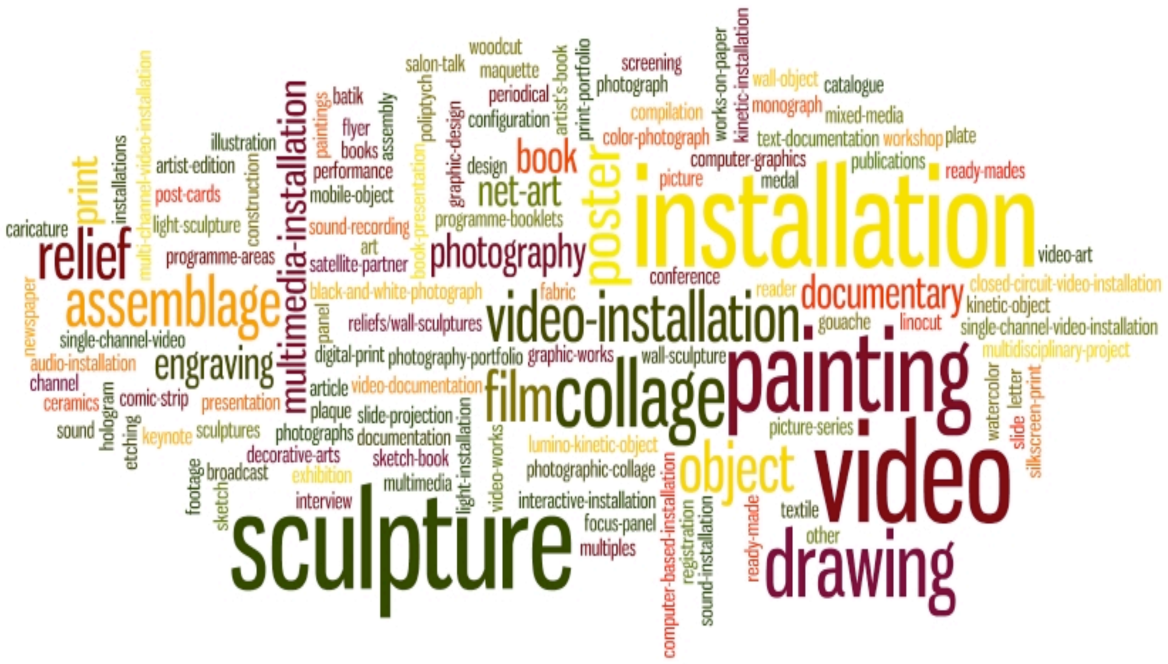


Figure 5: Tag cloud of used keywords

From this analysis, we extracted a set of terms that covers most of the partners' annotations for artworks. These terms are shown in the table below.

Term	AAT Identifier
Video art	(ID: 300102067)
Installations (visual works)	(ID: 300047896)
Sculpture (visual work)	(ID: 300047090)
Paintings (visual works)	(ID: 300033618)
Drawings (visual works)	(ID: 300033973)
Collages (visual works)	(ID: 300033963)
Posters	(ID: 300027221)
Found objects	(ID: 300047210)
Motion pictures (visual work)	(ID: 300136900)
Net art	(no ID)
Photographs	(ID: 300046300)
Prints (visual works)	(ID: 300041273)
Artists' books	(ID: 300123016)
Generative art	(ID: 300266042)
Music	(ID: 300054146)
Sound art	(ID: 300047267)
Performance art	(ID: 300121445)
Television programs	(ID: 300263432)
Documents	(ID: 300026030)

Table 4: Terms of the DCA vocabulary

The terms are actually selected from the AAT vocabulary (22). This way, each term of our DCA vocabulary gets an equivalent term in AAT. This wasn't possible for all terms. The term 'Net Art' doesn't have an equivalent in the AAT vocabulary. By binding the vocabulary, which enriches contemporary art descriptions, to the AAT, it is enriched itself, adding more context to the DCA vocabulary.

4.3 *Vocabulary Design*

The first thing that needs to be done is to assign URIs to the terms. In a Linked Open Data context, these URIs should preferably be HTTP URIs. At the same time, the identifiers need to be persistent to provide sustainability for the identification of resources. Persistent identifiers must provide access to the resources, even when their location changes over time.

There exist several standards for defining URIs (23). The most important ones are:

- Purl:
  Persistent Uniform Resource Locators[27] are URLs that redirect to the appropriate URL for a resource.
- URN:
  Uniform Resource Names[28] are designed to name a resource instead of denoting a locator for the resource, e.g., ISBN numbers for books.
- DOI:
  Digital Object Identifiers (DOI[29]) are a mechanism for permanent identification of electronic documents.
- Handles:
  Handles[30] are a partially bundled service (including protocols, a namespace, and a software implementation) where the creation and maintenance of identifiers and bindings are “outsourced” to a local repository hosting a Handle server. The central Handle service identifies the local server and directs the request to the server for resolution.
- OpenURL:
  OpenURL[31] is a URL with embedded metadata, which enables users to find the copy of a resource more easily. The metadata is used by the resolver service. It is often bibliographic in nature, and OpenURLs are commonly used by libraries.
- ARK:
  Ark[32] is a URL scheme, which can identify both physical and digital objects. Like PURLs, there is no added service cost for implementing and maintaining ARKs and no service dependencies. Like DOIs and Handles, ARKs theoretically exist independently of HTTP and DNS protocols, but unlike the former, currently they are only capable of action within them.

For our DCA vocabulary, we will use Purl. Our Purl is http://purl.org/DCAVocabulary/. The terms of our DCA vocabulary will have the following URL form: http://purl.org/DCAVocabulary/{vocabulary term}.

[27] http://purl.oclc.org/docs/index.html
[28] http://www.ietf.org/rfc/rfc2141.txt
[29] http://www.doi.org/
[30] http://www.handle.net/
[31] http://www.niso.org/kst/reports/standards?step=2&project_key=d5320409c5160be4697dc046613f71b9a773cd9e
[32] https://confluence.ucop.edu/display/Curation/ARK

These terms need to be added to a SKOS concept scheme. The scheme will be denoted by http://purl.org/DCAVocabulary/DCAscheme. Thus, first we created our concept scheme and then, we modelled all the terms as top-level concepts that will be added to the concept scheme. It was a design decision to model all the terms as skos:TopConcept. We started with a very basic vocabulary, to support multilingual search in Europeana. At the moment, this vocabulary can't be used as a classification scheme. For this purpose, the vocabulary is too concise. As shown in the previous chapter, a SKOS vocabulary can evolve over time, and remain compatible with its previous versions. In time, this vocabulary can be refined and then used as a classification scheme for contemporary art. Modelling a classification scheme for contemporary art was not feasible in the DCA project.

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix dcavoc: <http://purl.org/DCAVocabulary/> .
@prefix skos : <http://www.w3.org/2004/02/skos/core#> .

dcavoc:DCAscheme              rdf:type                skos:ConceptScheme;
                              skos:hasTopConcept      dcavoc:Video_Art;
                              skos:hasTopConcept      dcavoc:Installations;
                              skos:hasTopConcept      dcavoc:Sculpture;
                              skos:hasTopConcept      dcavoc:Paintings;
                              skos:hasTopConcept      dcavoc:Drawings;
                              skos:hasTopConcept      dcavoc:Collages;
                              skos:hasTopConcept      dcavoc:Posters;
                              skos:hasTopConcept      dcavoc:Found_Objects;
                              skos:hasTopConcept      dcavoc:Motion_Pictures;
                              skos:hasTopConcept      dcavoc:Net_Art;
                              skos:hasTopConcept      dcavoc:Photographs;
                              skos:hasTopConcept      dcavoc:Prints;
                              skos:hasTopConcept      dcavoc:Artists_Books;
                              skos:hasTopConcept      dcavoc:Generative_Art;
                              skos:hasTopConcept      dcavoc:Music_Art;
                              skos:hasTopConcept      dcavoc:Sound_Art;
                              skos:hasTopConcept      dcavoc:Performance_Art;
                              skos:hasTopConcept      dcavoc:Television_Programs;
                              skos:hasTopConcept      dcavoc:Documents .
```

Now, we have our concept scheme, representing our DCA SKOS vocabulary. We still need to define all our top concepts. Below, we give an example of how such a skos:TopConcept looks in the DCA vocabulary.

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix dcavoc: <http://purl.org/DCAVocabulary/> .
@prefix skos : <http://www.w3.org/2004/02/skos/core#> .

dcavoc:Video_Art              rdf:type                skos:Concept;
                              skos:prefLabel          “Video Art”@en;
                              skos:prefLabel          “Video Kunst”@nl;
                              skos:prefLabel          “Videokunst”@de;
                              skos:prefLabel          “l'Art Vidéo”@fr;
                              skos:prefLabel          “ Vídeó List”@is;
                              skos:prefLabel          “ Video Umetnost”@sl;
                              skos:prefLabel          “Video Arte”@es;
                              skos:prefLabel          “Video Arte”@pt;
                              rdf:ID                  300102067.
```

For now rdf:ID is used to denote the AAT identifier of the terms. There are plans to publish the AAT vocabulary as Linked Open Data. Once, this has been done, we can enrich our DCA vocabulary properly using owl:sameAs instead of using rdf:ID to denote the AAT's identifier for the term.

In a later phase, the DCA vocabulary may obtain more terms and hierarchy, without affecting the already annotated content. Updates of the DCA vocabulary must be done carefully, to guarantee its backward compatibility.

5 Bibliography

1. *RDF Primer.* [Online] http://www.w3.org/TR/rdf-primer/.
2. *RDF/XML Syntax Specification (Revised).* [Online] http://www.w3.org/TR/rdf-syntax-grammar/.
3. **Tim Berners-Lee, Dan Connolly.** *Notation3 (N3): A readable RDF syntax.* s.l. : W3C, 2008.
4. *Turtle - Terse RDF Triple Language.* [Online] http://www.w3.org/TeamSubmission/turtle/.
5. *Linked Data - The Story so far.* **Christian Bizer, Tom Heath, and Tim Berners-Lee.** 3, s.l. : International Journal on Semantic Web and Information, 2009, Vol. 5.
6. **Ben Adida, Ivan Herman, Manu Sporny, Mark Birbeck.** *RDFa 1.1 Primer.* 2012.
7. **Eric Prud'hommeaux, Andy Seaborne.** *SPARQL Query Language for RDF.* 2007.
8. *RDF Vocabulary Description Language 1.0: RDF Schema.* [Online] http://www.w3.org/TR/rdf-schema/.
9. *OWL Web Ontology Language .* [Online] http://www.w3.org/TR/owl-ref/.
10. **Antoine Isaac, Ed Summers.** *SKOS Simple Knowledge Organization System Primer.* 2009.
11. *Europeana Data Model.* [Online] http://group.europeana.eu/c/document_library/get_file?uuid=718a3828-6468-4e94-a9e7-7945c55eec65&groupId=10605.
12. **Dow, Martin.** *The OpenART Linked Events Model.* 2010.
13. *Definition of the CIDOC Conceptual Reference Model.* [Online] http://www.cidoc-crm.org/docs/cidoc_crm_version_5.0.3.pdf.
14. **Dan Brickley, Libby Miller.** *FOAF Vocabulary Specification 0.98.* 2010.
15. **WonSuk Lee, et. al.** *Ontology for Media Resources 1.0.* s.l. : W3C, 2012.
16. *PREMIS Data Dictionary for Preservation Metadata.* [Online] http://www.loc.gov/standards/premis/v2/premis-dd-2-1.pdf.
17. **Brickley, Dan.** *Basic Geo (WGS84 lat/long) Vocabulary.* s.l. : W3C, 2004.
18. **Ryan Shaw, Raphaël Troncy, Lynda Hardman.** *LODE: An ontology for Linking Open Descriptions of Events.* 2010.
19. **Hepp, Martin.** *GoodRelations Language Reference.* 2011.
20. Dublin Core Metadata Initiative Homepage. [Online] http://dublincore.org/.
21. *Ontology:DOLCE+DnS Ultralite.* s.l. : ontologydesignpatterns.org, 2006.
22. *Art & Architecture Thesaurus.* [Online] http://www.getty.edu/research/tools/vocabularies/aat/about.html.
23. **Gordon McKenna, Roxanne Wyms.** *persistent identifiers (pids):recommendations for institutions.* s.l. : Athena.
24. *HTML Specification.* [Online] http://www.w3.org/TR/html401/.
25. **Carl Lagoze, Herbert Van de Sompel, Michael Nelson, Simeon Warner.** *The Open Archives Initiative Protocol for Metadata Harvesting.* 2008.
26. *LIDO - Lightweight Information Describing Objects.* [Online] http://www.lido-schema.org/schema/v1.0/lido-v1.0-specification.pdf.
27. **Kay, Michael.** *XSL Transformations (XSLT) Version 2.0.* 2007.

Appendix: Milestone deliverable: LOD made practical

DELIVERABLE

Project Acronym: DCA

Grant Agreement number: 270927

Project Title: Digitising Contemporary Art

D5.3 Enrichment module and POC

Revision: V1.1

Author(s): Sam Coppens (IBBT/MMLab); Erik Mannens (IBBT/MMLab)
Reviewer(s): Rony Vissers (PACKED vzw); Joris Janssens (PACKED vzw)

Project co-funded by the European Commission within the ICT Policy Support Programme		
Dissemination Level		
P	Public	X
C	Confidential, only for members of the consortium and the Commission Services	

REVISION HISTORY AND STATEMENT OF ORIGINALITY

Revision History

Revision	Date	Author	Organisation	Description
V0.1	31/08/2012	SC, EM	IBBT	First draft
V0.2	03/09/2012	RV, JJ	PACKED vzw	First internal review
V0.3	13/09/2012	SC, EM	IBBT	Amended draft
V1.0	13/09/2012	RV, JJ	PACKED vzw	Final review
V1.1	17/01/2013	RV	PACKED vzw	Proofreading by native English speaker

Statement of originality:

This deliverable contains original unpublished work except where clearly indicated otherwise. Acknowledgement of previously published material and of the work of others has been made through appropriate citation, quotation or both.

Content Table

Executive Summary

In this deliverable, we discuss how to publish data as Linked Open Data in practice. This means we show how to do it, which steps to take and which open source tools can be used for the different steps.

The basic steps in publishing data as Linked Open Data are:

- select appropriate RDF model to publish the data;
- choose a Linked Open Data server infrastructure;
- transform the data to RDF;
- enrich the data.

Each of the steps is discussed in detail and demonstrated by our proof of concept. An important step in publishing Linked Open Data is to enrich the data. In this deliverable, special attention will be given to the enrichment module.

Finally, we'll present this deliverable with a workflow recommendation for the DCA partners to publish their data as Linked Open Data.

1. Linked Open Data publication

Introduction

The main goal of Linked Open Data (LOD) (1) is to allow people to share structured data on the Web as easily as they share documents today. It actually refers to a style of publishing and interlinking structured data on the Web. The main recipe for LOD is RDF (2), the Resource Description Framework. The structured data is published as RDF data (using an RDF data model) and RDF links are used to link data from different data sources. This way, the LOD on the Web creates a giant global graph, across which all the published data is connected to each other. It's also called the Web of Data. Clients can easily discover and consume data through it.

LOD stipulates four basic principles [1]:

- The first principle is that we have to identify the items of interest in our domain first of all. Such items are the resources, which will be described in the data.
- The second principle is that those resources have to be identified by HTTP URIs and one should avoid schemes such as Uniform Resource Names (URNs) and Digital Object Identifiers (DOIs).
- The third principle is to provide useful information when accessing an HTTP URI.
- The fourth rule is to provide links to the outside world, i.e. to connect the data with that from other datasets in the Web of Data. This makes it possible to browse data from a certain server and receive information from another server. In other words, by linking the data with that of other datasets, the web becomes one huge database, called the Web of Data.

To demonstrate LOD publication, a proof of concept is set up. This is provided by an LOD server which publishes and enriches the DCA content. The URL of the LOD server is http://dca.test.ibbt.be:8080/. At present, the LOD server puts some example records online as EDM records. In the future, the DCA content will be harvested, enriched and published as LOD, as soon as *Europeana* has harvested all the content.

Preliminaries / requirements

When publishing structured data as LOD, first of all we need to select the items, which will be published as LOD. These resources will be identified using HTTP URIs, so that the resources become available on the Web using standard HTTP mechanisms (e.g., using a web browser). These two conditions comply with the first two principles for publishing LOD.

The selected and identifiable resources need of course to be described. For this, as explained earlier, we use RDF models. This is a first preliminary for publishing LOD: a metadata model for publishing the selected resources as RDF to the Web. This refers to the third principle of publishing LOD.

Every described resource can have multiple representations, e.g., HTML (3), RDF/XML (4), Turtle (5), etc. These are there to provide useful information on the resource. Assuming that the client is a human using a web browser to look up some information about a resource, the server publishing the LOD will then serve an HTML page, because HTML is designed to present information to a human. When a machine agent wants to look up information about the resource, the server will provide an RDF/XML representation about the resource, because RDF/XML is machine-readable. This content

[1] http://www.w3.org/DesignIssues/LinkedData.html

negotiation is the responsibility of the LOD server that will serve the right representation about a resource, based on the preferences of the client. This also refers to the third principle of LOD.

In order to provide links to external data sources, the fourth principle of publishing LOD, the published content needs to be enriched, via an enrichment module. This is a fourth preliminary for publishing LOD, which will interconnect all the published LOD content in order to create the Web of Data.

To summarise, the following entities are preliminary for LOD publication:

- a metadata model;
- an LOD server;
- an enrichment module.

These three entities will be discussed in detail in the next sections.

2. Linked Open Data Model

Purpose

When publishing data on the Web, you need a data model to describe your data. The same holds true for publishing LOD, which relies on RDF models to describe the data. In RDF models, everything is described using triples, as explained in more detail in deliverable 3.1 *Metadata implementation guidelines for digitised contemporary artworks* and deliverable 3.2 *Recommendation on contextualisation and interlinking digitised contemporary artworks*. These models are targeted at making links. One can easily interconnect pieces of information, each using their own RDFS (6) / OWL (7) ontology. One can even mix the ontologies to describe your pieces of information.

When developing a data model, one should choose those ontologies that best fit one's needs or one can make one's own schema, but then one needs to link it to popular ontologies to make one's data more interoperable. There exist various ontologies, each targeted at describing a specific thing or piece of information. In the next section, we provide examples of ontologies that are well known and much used, classified by their object of description.

RDF/RDFS/OWL ontologies

Cross-domain

Dublin Core (DC, (8)): a very generic model that can describe basic features (who, what, where, and when) about anything. The core model exists of fifteen properties.

Dolce Ultra Lite (DUL, (9)): a lightweight upper ontology that describes general concepts across all knowledge domains. Such upper ontology is often used to unify data, described using various ontologies.

Persons/organisations

Friend-Of-A-Friend (FOAF, (10)): a vocabulary for describing persons and organisations, e.g., the first and last name of a person, their social network accounts, contact information, etc.

Locations

Basic Geo (WGS84 lat/long) Vocabulary (11): a very basic vocabulary to express the longitude and latitude of a location. It is very simple but widely spread and powerful in combination with Dublin Core or FOAF.

Events

Linking Open Description of Events (LODE, (12)): a basic model that can describe events. It describes the event itself, when it took or will take place, where, which actors are involved, etc.

Cultural heritage

Europeana Data Model (EDM, (13)): a model that is targeted at describing cultural heritage information as LOD. It is developed by and for *Europeana* and is closely related to the LIDO schema.

OpenART (14): an event-driven ontology produced to describe 'art world' datasets. The ontology is split into a number of parts to allow greater re-usability. It's linked to the DUL ontology for greater applicability and interoperability.

Products

GoodRelations (15): a model that describes products and services offered for e-commerce. It is able to describe all the details of the products, such as price, stock, etc.

Proof of concept, developed within the framework of DCA

For the proof of concept, the EDM model is employed to publish DCA data on contemporary artworks. The EDM model was chosen, because it was especially designed by *Europeana* to publish their content on cultural heritage as LOD.

To ingest the data, the proof of concept relies on OAI-PMH (16) for harvesting. OAI-PMH is a harvesting protocol, which is supported by many aggregators and platforms of cultural heritage institutions. The OAI-PMH protocol was discussed in detail in deliverable 5.1 *Assessment of the different aggregation platforms and their aggregation requirements*.

Many DCA content partners will provide their data to *Europeana* using the MINT tool[2] for mapping and harvesting. From this MINT platform, we can ingest the data. The formats we accept using OAI-PMH are EDM and LIDO (17). When LIDO records are received, they are mapped to EDM records using an XSLT (18) mapping schema. For this XSLT, we followed EDM mapping guidelines, published by *Europeana* Professional[3]. Only the metadata are ingested into the proof of concept, no Web resources (digital representations).

When mapping to an RDF model that needs to be published as LOD, attention needs to be given to the used URIs. These all need to be HTTP URIs. Every instance of an RDF class needs to have an HTTP URI. Once an URI has been established the content negotiation can take place, when the data is published as LOD. This content negotiation is discussed in detail in the next section. For the form of URIs we have chosen for the following composition:

[BASE URI LOD Server (http://dca.test.ibbt.be:8080)]/resource/[class name]/ID

In EDM the following class instances are present:

Core EDM classes

• edm:ProvidedCHO	The provided cultural heritage object.
• edm:WebResource	The web resource that is a digital representation.
• edm:Aggregation	The aggregation that groups classes together.

Contextual EDM classes

• edm:Agent	The related agents (persons, organisations).
• edm:Place	The related locations.
• edm:Timespan	The related timespans.
• skos:Concept	The related SKOS (19) concepts.

Apart from the SKOS instances, which are hosted elsewhere, all the instances of these classes need HTTP URIs in the form shown above.

[2] http://mint.image.ece.ntua.gr/redmine/projects/mint/wiki
[3] http://pro.europeana.eu/documents/900548/ea68f42d-32f6-4900-91e9-ef18006d652e

3. LOD server

Purpose

Once we have the RDF model and instances of it, we need to store this metadata and need to publish it as LOD. The LOD server is responsible for publishing the information. This means it is responsible for giving access to the published data (via standardised HTTP methods) and for querying it. Both functions of the LOD server are discussed in detail in the sections below.

Access

As explained earlier, the LOD server needs to provide useful information to the client. This information on the published resources is represented using RDF. A normal web browser should not serve raw RDF to an HTML browser. Therefore, it will serve, alongside the RDF representation of the information, an HTML one. As such, the LOD server should serve representations of the data that can be understood by both humans (HTML) and machine agents (RDF). To achieve this, there are two main approaches:

- Embedding RDFa (20) in the HTML: In this approach, the server serves only one representation of the information, i.e., the HTML representation. The RDF data is embedded within it, using RDFa. RDFa is a microformat for capturing RDF information.
- Content Negotiation: Here, the LOD server will serve two representations of the published data: the HTML representation and the RDF one. The HTTP client sends an HTTP header in each request in which the client indicates which representation he prefers. Based on this preference, the content negotiation can take place. In practice, this means that every resource described by an RDF scheme has to be identified by an HTTP URI, (e.g., http://dbpedia.org/resource/Gerhard_Richter). Every resource should also have two representations: an XHTML (human-readable) and an RDF representation (machine-readable). Every representation also has to be identified by an HTTP URI (e.g. http://dbpedia.org/page/Gerhard_Richter) for the XHTML representation and for RDF representation (e.g. http://dbpedia.org/data/Gerhard_Richter). When coming across the HTTP URI of a resource, the LOD server determines which representation should be served, based on information in the accept header of the user's client, and then redirects the client to the appropriate representation using 303 redirect and content negotiation. This is shown in figure 1.

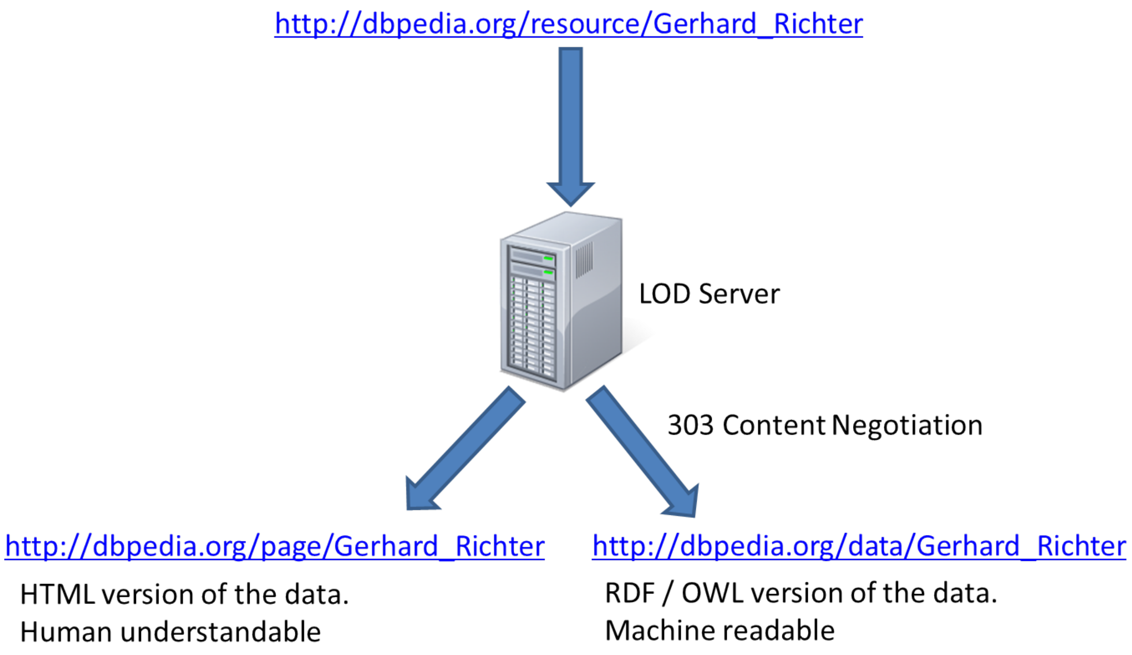

http://dbpedia.org/resource/Gerhard_Richter
LOD Server
303 Content Negotiation
http://dbpedia.org/page/Gerhard_Richter
HTML version of the data.
Human understandable
http://dbpedia.org/data/Gerhard_Richter
RDF / OWL version of the data.
Machine readable

Figure 1: content negotiation LOD server

Query

Another task for the LOD server is to make sure the content it publishes can be queried. SPARQL (21) was designed specifically to this end. SPARQL is a query language and data access protocol for the Semantic Web. It is defined in terms of the W3C's RDF data model and will work for any data source that can be mapped into RDF. As such, the LOD server is also responsible for publishing a SPARQL endpoint, at which SPARQL queries can be fired and results are returned.

Providing a SPARQL endpoint as LOD server is very important. It opens up your data and lets other data sources use it to enrich their own. Of all query languages, those that emulate SQL syntactically have probably been the most popular and widely implemented. This is perhaps surprising given the very different models that lurk behind relational databases and RDF. Familiarity with syntax has no doubt contributed to such success. SPARQL follows this well-trodden path, offering a simple, reasonably familiar (to SQL users) SELECT query form.

The SPARQL query below returns the names and emails of every person whose information is published by a LOD server. The persons are described using FOAF ontology. Notice the similarity with SQL, except for the WHERE clauses, which are formulated using RDF.

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name ?email
WHERE
{
	?person a foaf:Person.
	?person foaf:name ?name.
	?person foaf:mbox ?email.
}
```

Listing 1: example SPARQL query

Open source tools

There exist a lot of open source frameworks which provide an LOD server. These are targeted at publishing the data and to providing a SPARQL endpoint for the data. In this section, we provide a list of open source tools that can be used for putting up an LOD server.

The tools are categorised according to purpose. We make a distinction between triple stores, Linked Data frontends, SPARQL endpoints and relational database LOD publishers. There are two practical ways of publishing RDF data as LOD:

- One can take an out-of-the-box solution: These platforms provide a triple store for storing your RDF data, a SPARQL endpoint to query the RDF data over HTTP, and a Linked Data frontend to publish RDF data using content negotiation. An example of such a platform is Openlink Virtuoso.
- One can compile one's own LOD server: For this one must first choose a triple store for storing the RDF data. Most of the triple stores just provide storage for your RDF data. They can all handle SPARQL queries, but this doesn't mean they make an SPARQL endpoint available over HTTP. Once one has a triple store one needs a server to set up a SPARQL endpoint over HTTP and a Linked Data frontend to publish data as LOD.

Triple Stores

Triple stores are RDF databases. They store the RDF data and provide a SPARQL endpoint for the data. Most do not offer public access to the data. This means they do not publish the data; they only store it and make sure it can be queried using SPARQL. This means that you have to transform your data to RDF yourself and provide a Linked Data frontend yourself, which will do the content negotiation.

Jena[4]

Jena is a Java RDF API and toolkit. It is a Java framework to construct semantic web applications. It allows you to store data in-memory, and to query the data. It also allows one to reason over the data using RDFS or OWL. Jena is no server that publishes your data or provides a SPARQL endpoint.

Sesame[5]

Sesame is very similar to Jena. It is a Java RDF database, with support for RDFS reasoning and SPARQL querying. It has a wide range of tools for developers.

TDB[6]

TDB is part of Jena and provides persistent RDF storage and query. TDB can be used as a high performance RDF store on a single machine. TDB is not a server, nor does it provide a SPARQL endpoint.

Openlink Virtuoso (open source edition)[7]

Openlink Virtuoso is a general-purpose database. It is actually relational and also provides a Linked Data interface and a SPARQL endpoint for the data. It is also a server, publishing data as LOD and providing a SPARQL endpoint.

OWLIM[8]

[4] http://jena.apache.org/
[5] http://www.openrdf.org/
[6] http://jena.apache.org/documentation/tdb/index.html
[7] http://virtuoso.openlinksw.com/dataspace/dav/wiki/Main/

OWLIM is a very scalable triple store. It also provides an inference engine and SPARQL query engine.

Linked Data frontends

Linked Data frontends usually take their data from an SPARQL endpoint and provide a Linked Data interface to the data from it. These frontends take care of the content negotiation needed. Some triple stores are also servers and already provide a Linked Data interface, e.g., Openlink Virtuoso, while others don't, e.g., TDB or Jena.

Pubby[9]

Pubby can be used to provide a Linked Data interface to SPARQL endpoints. When you already have your data in a triple store, Pubby provides the Linked Data interface (with content negotiation).

Linked Data Pages[10]

Linked Data Pages is a Linked Data publishing framework in PHP. It relies, just like Pubby, on a SPARQL endpoint for publishing the data.

SPARQL endpoints

Triple stores that are also servers. They mostly provide a SPARQL endpoint already. Some triple stores, e.g., Jena or Sesame, do not provide this functionality. These tools will implement a SPARQL endpoint. They mostly have a binding to the data in the form of a model, which is implemented by Jena.

SPARQLer[11]

SPARQLer is a general-purpose SPARQL processor and query validator, based on Jena. It can be used to provide a local SPARQL endpoint.

Joseki[12]

Joseki is an HTTP engine that supports SPARQL. It can be used to put up SPARQL endpoints.

Fuseki[13]

Fuseki is a SPARQL server. It replaces Joseki and provides REST-style SPARQL support. It can be easily used in combination with Jena or TDB.

Su4j[14]

Su4j is a Jena-based servlet implementation of SPARQL. It can be used to provide a SPARQL endpoint.

---

[8] http://www.ontotext.com/owlim
[9] http://www4.wiwiss.fu-berlin.de/pubby/
[10] https://github.com/csarven/linked-data-pages
[11] http://sparql.org/
[12] http://www.joseki.org/
[13] http://jena.apache.org/documentation/serving_data/index.html
[14] https://bitbucket.org/fundacionctic/su4j/wiki/Home

Relational database LOD publishers

When your data comes from a relational database, there exist solutions that put an extra layer on top of the database to provide a Linked Data interface and SPARQL endpoint. This way, the data from your relational database doesn't need to be transformed and migrated to a triple store. A disadvantage of this method is that such a triple store is not that flexible in storing data using various ontologies, because they have to be reflected in the database structure.

*D2R Server*15

D2R Server is a tool for publishing the data from relational databases on the semantic Web. The tool provides a SPARQL endpoint for the data and a Linked Data interface.

Openlink Virtuoso

Openlink Virtuoso is actually a relational database with an extra layer for publishing and querying linked data from relational databases, as explained above.

Proof of concept, developed within the DCA framework

For the proof of concept within the DCA framework, we made our own LOD server, based on tools described above. The LOD server consists of the following components:

- an application server;
- a triple store;
- a SPARQL endpoint;
- a Linked Data frontend.

The figure below gives an overview of our architecture for publishing the LOD. Each of the following components is discussed in detail in the following section.

[15] http://d2rq.org/d2r-server

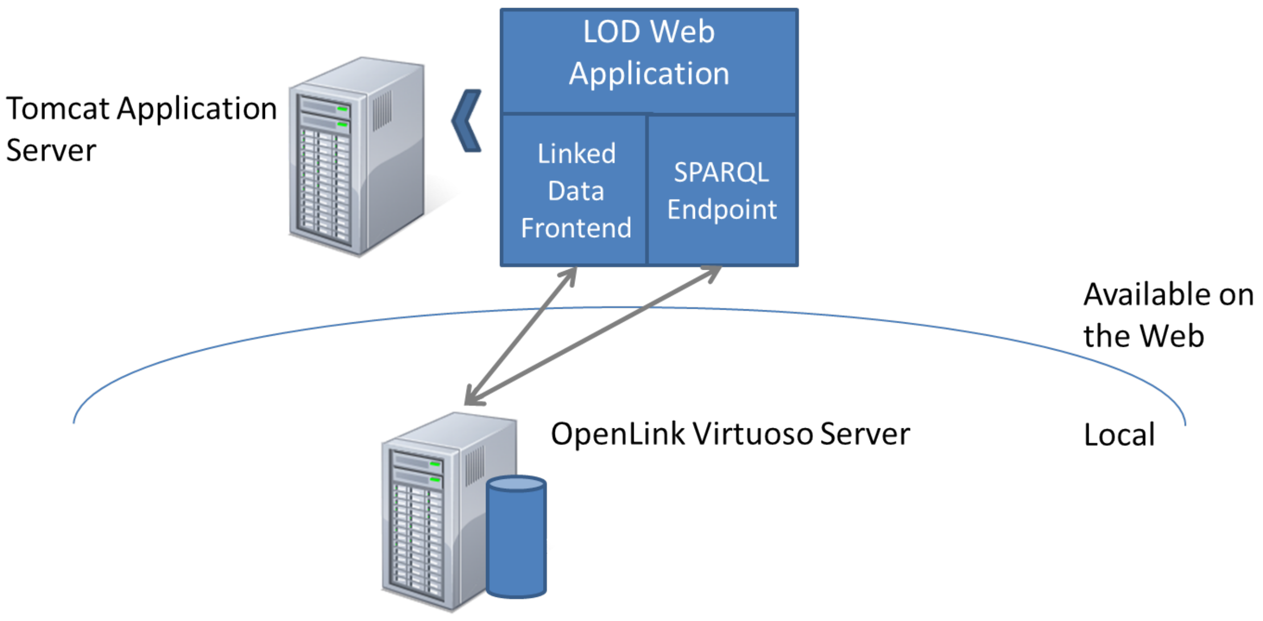


Figure 2: LOD server architecture of the DCA POC

Application server

We have chosen Apache Tomcat[16] as application server. It is an open source application server, implementing Java Servlet and JavaServer Pages technologies. It is easy to use and has a huge user community.

Triple store

Our triple store choice is Openlink Virtuoso. Virtuoso Universal Server is a middleware and database engine hybrid that combines the functionality of traditional RDBMS, ORDBMS, virtual database, RDF, XML, free-text, web application server and file server functionality in a single system. Virtuoso is as such a universal server. The open source edition of Virtuoso Universal Server is also known as OpenLink Virtuoso. Based on the BSBM benchmark, Openlink Virtuoso offers a good scalability and query performance. It is just used for storing the RDF data.

SPARQL endpoint

Openlink Virtuoso is also a server and offers a SPARQL endpoint over HTTP directly. However, we've developed our own SPARQL endpoint. This way, the Virtuoso server could stay local and only the web application publishing LOD is opened up for the Web. At the same time, we can maintain more control over the resources that can be queries using the SPARQL endpoint.

Joseki was used to implement the SPARQL endpoint. Joseki just needs to make a connection to the triple store in order to retrieve the RDF data model and then publish a SPARQL endpoint for it. It is a means of keeping control of what is in the model for the SPARQL endpoint and what is not.

[16] http://tomcat.apache.org/

Linked Data frontend

In the proof of concept, we made our own Linked Data frontend. To create our own Linked Data interface, we implemented three servlets. One was for content negotiation, one for providing HTML representation of the data, and the third one was to deliver the RDF model in the asked representation (RDF/XML, n-tuples or json). The Figure below gives a schematic overview of the servlets and a piece of example code is shown for content negotiation.

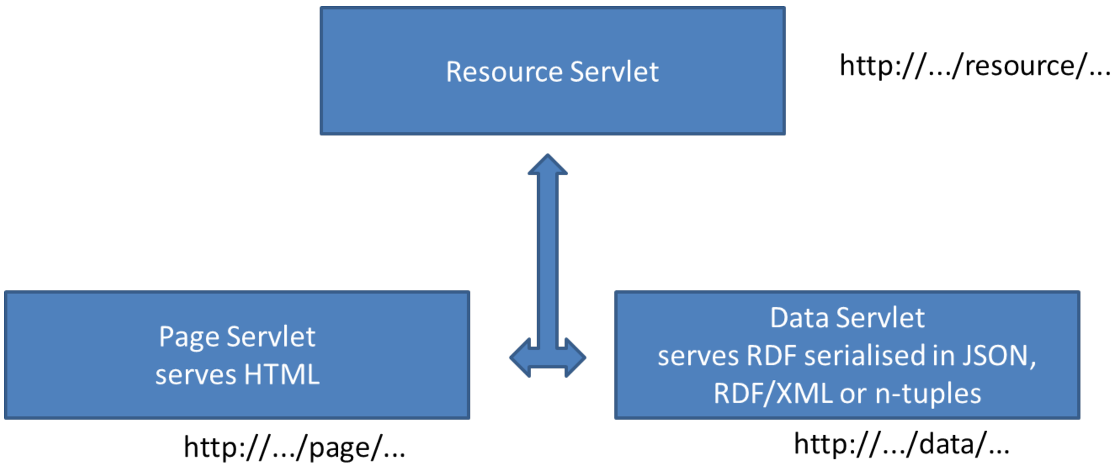


Figure 3: content negotiation and URI templates DCA PoC

Example code for the Resource servlet to do the content negotiation:

```
import java.io.IOException;

import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import de.fuberlin.wiwiss.pubby.negotiation.ContentTypeNegotiator;
import de.fuberlin.wiwiss.pubby.negotiation.MediaRangeSpec;
import de.fuberlin.wiwiss.pubby.negotiation.PubbyNegotiator;

public class ResourceServlet extends HttpServlet {

      public void doGet(HttpServletRequest request, HttpServletResponse response)
                  throws IOException {
            String relativeResourceURI = request.getRequestURI().substring(
                        request.getContextPath().length()
                              + request.getServletPath().length());
            // Some servlet containers keep the leading slash, some don't
            if (!"".equals(relativeResourceURI)
                        && "/".equals(relativeResourceURI.substring(0, 1))) {
                  relativeResourceURI = relativeResourceURI.substring(1);
            }
            if (request.getQueryString() != null) {
                  relativeResourceURI = relativeResourceURI + "?"
                              + request.getQueryString();
            }

            response.addHeader("Vary", "Accept, User-Agent");
            ContentTypeNegotiator negotiator =
PubbyNegotiator.getPubbyNegotiator();
```

```
            MediaRangeSpec bestMatch = negotiator.getBestMatch(request
                        .getHeader("Accept"), request.getHeader("User-Agent"));
            if (bestMatch == null) {
                  response.setStatus(406);
                  response.setContentType("text/plain");
                  response.getOutputStream().println(
                              "406 Not Acceptable: The requested data format is
not supported. "
                                          + "Only HTML and RDF are available.");
                  return;
            }

            response.setStatus(303);
            response.setContentType("text/plain");
            String location;

            if ("text/html".equals(bestMatch.getMediaType())) {
                  location = pageURL(relativeResourceURI); // method to generate
page URL from relativeResourceURI
            } else {
                  location = dataURL(relativeResourceURI); // method to generate
data URL from relativeResourceURI

            }
            response.addHeader("Location", location);
            response.getOutputStream().println(
                        "303 See Other: For a description of this record, see "
                                    + location);
      }
}
```

Listing 2: example Java code of content negotiation

4. LOD enrichment

Purpose

Enriching the data will actually interlink it with data from other data sources. These enrichments will therefore link your data graph to the giant, global data graph. Data can easily be discovered and consumed through it.

The figure below shows how the giant global data graph[17] looks today. Every dot in the graph is a dataset being published as LOD. The interconnections to the LOD datasets are enrichments. The content can be categorised. There is data on media (blue), geographic data (orange), data on publications (green), user generated data (red), government data (turquoise), cross-domain data (light blue) and life sciences data (purple).

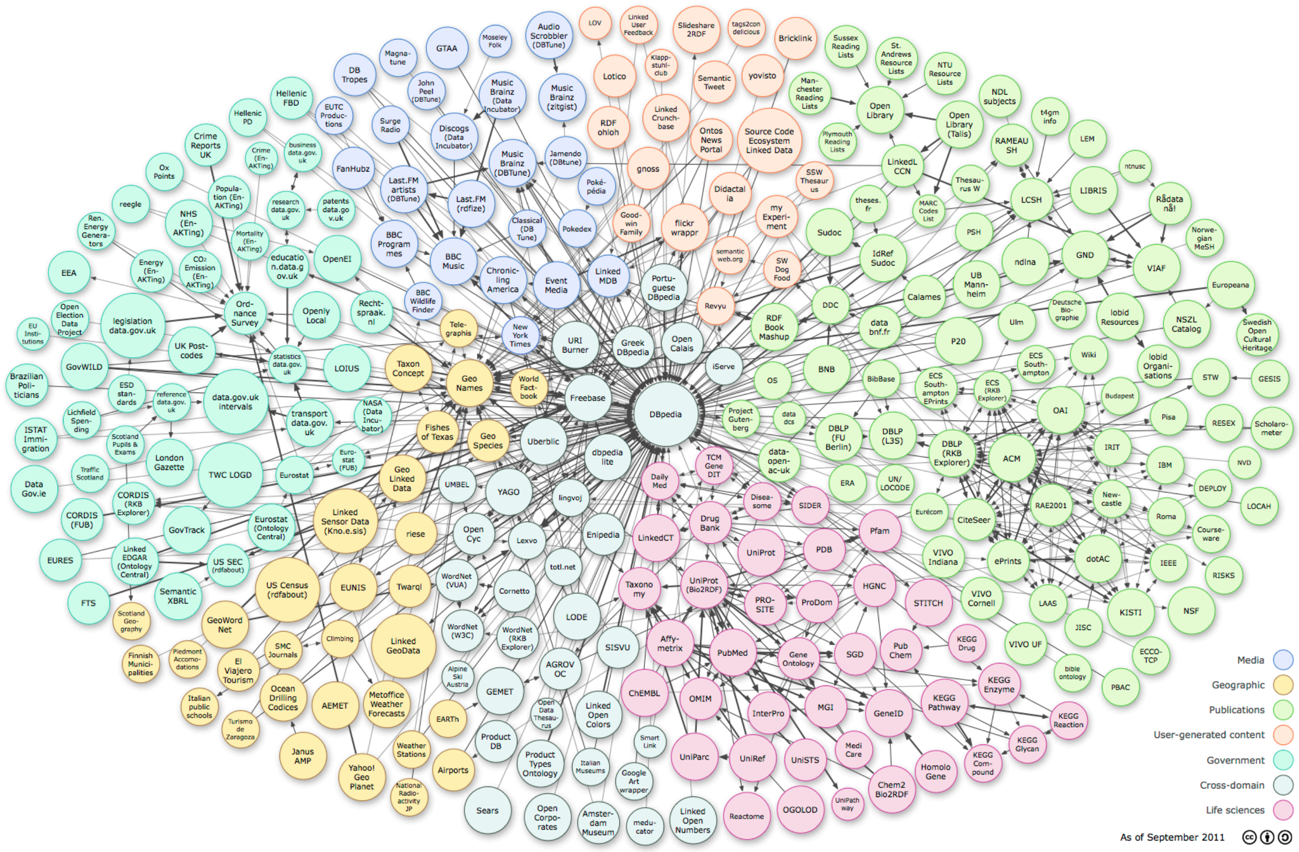


Figure 4: giant global data graph

The enrichments serve a dual purpose. Firstly, they help to disambiguate the data. If you have a resource describing a person called Steve McQueen, for example then this information is not enough to disambiguate Steve McQueen from Steve McQueen (the American actor), or Steve McQueen (the British visual artist / filmmaker). When the resource is linked to an external one, then this information helps to disambiguate it. When the resource is linked to http://viaf.org/viaf/9855712, we know we are talking about Steve McQueen, the American actor. When the resource is linked to http://viaf.org/viaf/96536223, the resource denotes Steve McQueen, the British visual artist / filmmaker.

[17] http://linkeddata.org/

A second purpose for interlinking data is to bring in some extra information about the resource. Let's look at the example of Steve McQueen, the British visual artist / filmmaker. If we only have a name and the year of birth for Steve McQueen in the triple store, then linking the resource to http://viaf.org/viaf/96536223 brings in extra information on him, such as titles of publications and unique identifiers from the Getty's Union List of Artist Names (www.getty.edu/research/tools/vocabularies/ulan/). The more your data is interlinked with external resources, the more value the data obtains and the more meaningful yours becomes. By enriching your data, you add more context information to it.

A key ingredient for discovering enrichments is SPARQL. Via a SPARQL endpoint the published datasets can be queried to detect enrichments. When publishing your data as LOD, a SPARQL endpoint is therefore crucial. It lets other data sources link to your dataset.

There exist various approaches for enriching data. You can do it manually, but this only holds for very small datasets. Bigger datasets ask for more automatic approaches. You can develop your own enrichment module, with your own specific enrichment algorithms, or you can use an enrichment tool, which will discover enrichments for you.

When developing your own enrichment module, the enrichment algorithm can be pattern-based or property-based. In the pattern-based approach, the enrichments are based on generally accepted naming schemas. A typical example of one such naming schema is the ISBN numbers. They uniquely identify publications/books. One can use them to search for remote records via SPARQL, which describe the same publication (publications with the same ISBN number). You can then link your records to the remote records discovered using the owl:sameAs property.

Another approach in developing your own enrichment algorithm is the property-based approach. With it, you can use certain properties of an entity to discover the same entity in a remote dataset. For instance, a person can't be uniquely identified by just his/her name. But the combination of the name, a date and place of birth will uniquely identify that person. You can use SPARQL to query remote datasets and look for persons with the same name, date and place of birth. Instead of relying on a unique naming convention to identify things, as in the pattern-based approach, you rely on a unique combination of properties to identify the same thing.

Open Source Tools

SILK[18]

SILK is a framework for discovering relationships between data items within different data sources. It is based on SPARQL. Via a configuration file, you can define your own SPARQL queries to discover relationships. These can be pattern-based or property-based.

Proof of concept, developed within the framework of DCA

For enriching the content of our PoC, we have developed our own property-based enrichment module. Our enrichment algorithm will make a distinction between the different types of entities, e.g., persons or locations, present in a record.

In D3.1 *Metadata implementation guidelines for digitised contemporary artworks* we gave some metadata best practices for content dissemination through *Europeana*. All created records from the

[18] http://www4.wiwiss.fu-berlin.de/bizer/silk/

contemporary art institutions within the DCA project are harvested and mapped to LIDO. *Europeana* can then harvest hese LIDO records. A list of properties for every sort of item that should be provided by the content partner was published in the deliverable. The tables below show the property lists for every sort of item, i.e., artworks, digital resources (surrogates), events, and actors. The values for fields denoted with an asterisk should be taken from a controlled vocabulary.

Artwork	Minimum	Recommended	Additional
ID	X		
Title	X		
Date		X	
Type	X		
Description		X	
Place		X	
Measurements			X
Collection		X	
Rights		X	
Language		X	
Subjects/keywords*	X		
Events			X
Surrogates			X

Surrogate	Minimum	Recommended	Additional
URL	X		
Description		X	
Language			X
Date		X	
Type*		X	
Rights		X	
Measurements			X
Format (mimetype)		X	

Event	Minimum	Recommended	Additional
Title	X		
Date		X	
Type*	X		
Description		X	
Place			X
Actor			X
Language		X	

Actor	Minimum	Recommended	Additional
Name	X		
Role*		X	
Place			X
Biography			X

Year of Birth		X	
Year of Death			X

Listing 2: table of recommended fields to be offered by the contemporary art institutions

These are also the properties used for interlinking data. This means that the properties will be used for building SPARQL queries. For every type of entity we want to enrich, we specify a property-based SPARQL query to look for enrichments. The entities we will be enriching are:

- locations;
- persons;
- artwork type;
- artwork.

For every type, we specify a set of properties to identify the entity uniquely. These properties can then be used to build queries for remote datasets. The datasets that are candidate for enrichment are discussed in the next Section.

Locations

Identifying properties:

- name;
- geographical coordinates.

Unfortunately, the geographical coordinates were not present in the content partners' databases. We can therefore only use the name of the location. Sometimes the country is also present in the content partners' databases. This property is used as extra context information.

```
PREFIX dbpediaprop: <http://live.dbpedia.org/property/dateOfBirth>
PREFIX dbpedia: <http://dbpedia.org/resource/ >
PREFIX dbpediaowl: <http://live.dbpedia.org/ontology/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT ?place
WHERE
{
?place a dbpediaowl:Place.
?place rdfs:label 'Rijeka'@en.
?place dbpediaowl:country dbpedia:Croatia.
}

Result: <http://dbpedia.org/resource/Rijeka>
```

Listing 3: example SPARQL query for DBpedia[19] looking for a place Rijeka

If the country is not specified, DBpedia could return more than one result. To solve this, we need human intervention to pick out the right one. This shows why it is important to deliver as much information as possible when data is being aggregated and published as LOD.

[19] http://dbpedia.org

Persons

Identifying properties:

- name;
- date of birth;
- place of birth.

```
PREFIX dbpediaprop: < http://live.dbpedia.org/property/dateOfBirth>
PREFIX dbpedia: <http://dbpedia.org/resource/ >
PREFIX dbpediaowl: <http://live.dbpedia.org/ontology/>
PREFIX foaf: < http://xmlns.com/foaf/0.1/>
SELECT ?person
WHERE
{
?person a foaf:Person.
?person dbpediaprop:name 'Gerhard Richter@en.
?person dbpediaprop:dateOfBirth '1932-02-09'^^xsd:date.
?person dbpediaowl:birthPlace dbpedia:Dresden.
}

Result: <http://dbpedia.org/resource/Gerhard_Richter>
```

Listing 4: example SPARQL query for DBpedia looking for a Gerhard Richter entity

The same holds true here, as with the places. If not all identifying properties are present in the record, the query will become less distinctive and more than one result could return. The solution is to have someone acting as an authority, picking out the right enrichment.

Artwork type

Identifying properties:

- The literal itself

The artwork type should actually refer to a term of a SKOS vocabulary. The SPARQL query below shows how to link the artwork type to a SKOS concept.

```
PREFIX skos: < http://xmlns.com/foaf/0.1/>
SELECT ?concept
WHERE
{
?concept a skos:Concept.
?concept skos:prefLabel 'Video Art'@en.
}
```

Listing 5: example SPARQL query looking for a concept named "Video Art"

Artwork

Identifying properties:

- Artwork ID

- Data provider

The artwork itself can be enriched with its *Europeana* equivalent, of course only if the artwork's metadata has already been delivered to *Europeana*. A combination of the ID and the data provider is sufficient to identify the *Europeana* equivalent. If *Europeana* supported SPARQL, the query would look like this:

```
PREFIX skos: < http://xmlns.com/foaf/0.1/>
SELECT ?artwork
WHERE
{
?artwork a ore:Aggregation.
?artwork edm:aggregatedCHO 'Hirst:1423'
?artwork edm:dataProvider 'MuZee'.
}
```

Listing 6: example SPARQL query for Europeana looking for an artwork of Hirst

In our PoC, the enrichment module is implemented in Java. It will query remote data sources looking for enrichments for each sort of entity (artwork, person, location, etc.). When querying the remote dataset, the identifying properties of the entities are used. Of course, not every DCA artwork description will contain all the information on identifying properties. In these cases, the query will be weakened using only the provided identifying properties of the entity. Such automatic enrichments are always subjected to many errors and human intervention is needed to ensure the found enrichment is correct. To solve this, the provenance is stored for each enrichment. This way, enrichments can be classified based on the use of identifying properties. Those using all the identifying properties are correct, those using only a subset of the identifying properties will need inspection to see if the found enrichment is still correct.

5. Enrichment sources

Purpose

In this section, we will elaborate on data sources that can be used to enrich one's data. These sources either make their data searchable via a SPARQL endpoint or via an API. These enrichment sources are classified based on the topic of their content.

Named entities

Named Entities are just known concepts, such as important events, places, persons, etc. Usually, named entities are extracted from free text. Once you have selected the named entities from a text, you can start using the other data sources to look for enrichments for them.

OpenCalais[20]

OpenCalais is the most famous named entity extractor. You can send text to OpenCalais and it will filter all the recognised named entities from the text. It can detect persons, locations, events, brands, music bands, etc. OpenCalais immediately links to DBpedia, Wikipedia, Freebase, Reuters.com, GeoNames, Shopping.com, IMDB, and Linked MDB. It provides web services which automatically annotate your content with rich semantic metadata. As well as its web services, OpenCalais also publishes its content as LOD. OpenCalais can only deal in English, French and Spanish.

DBpedia Spotlight[21]

DBpedia Spotlight is a named entity extractor like OpenCalais. It will annotate your free text with links to DBpedia resources. One drawback is that it is currently only available in English.

Cross-domain

These cross-domain datasets have information on various sorts of things, e.g., persons, locations, movies, books, cultural heritage, etc. They can almost always be used to enrich your data, no matter what domain your data belongs to. When looking at the giant global data graph, you will notice that these datasets are mostly used as a sort of interlinking hubs, which connect various other datasets to each other.

DBPedia[22]

DBpedia publishes the knowledge extracted from Wikipedia as LOD. It makes its data available via a SPARQL endpoint. It has information on persons, places, creative works (music albums, films, video games, etc.), organisations, species and diseases. DBpedia provides localised versions of DBpedia in 111 languages. It publishes its content as LOD and has a public SPARQL endpoint available for querying the dataset. DBpedia also provides RDF dumps, which can be downloaded and ingested into triple stores.

Freebase[23]

Freebase is very similar to DBpedia. It holds information on persons, locations, organisations, etc. The data from Freebase is collected from various data sources, such as ArXiv, CrunchBase, Eurostat, Wikipedia, IMDB, Library of Congress, etc. Freebase can take on queries using MQL, their

[20] http://www.opencalais.com/
[21] http://github.com/dbpedia-spotlight/dbpedia-spotlight
[22] http://dbpedia.org
[23] http://www.freebase.com/

own developed query language. They also provide a REST API and publish the content also as LOD. RDF dumps of Freebase are also available.

OpenCyc[24]

OpenCyc is the open source version of CYC. CYC is a very large general knowledge base (including a reasoning engine). The CYC ontology contains hundreds of thousands of terms and millions of assertions, relating the terms to each other, forming an (English) upper ontology of which all of human consensus reality domain is the domain. OpenCyc provides an API for application development.

YAGO2[25]

YAGO2 is a large semantic knowledge base derived from Wikipedia, WordNet, and GeoNames. It contains knowledge on 10 million entities (persons, organisations, cities, etc.). YAGO2 provides dumps, but it can also be queried and browsed.

Vocabularies

WordNet[26]

WordNet is a large lexical database of English. It organises nouns, verbs, adjectives, and adverbs in sets of cognitive synonyms (synsets), each expressing a distinct concept. WordNet superficially resembles a controlled vocabulary. WordNet can be browsed and is also available as dump.

LCSH[27]

Library of Congress Subject Headings are a thesaurus of subject headings. These subject headings capture the essence of the topic of a document. The thesaurus can be used as controlled vocabulary to classify bibliographic records. LCSH are currently available as LOD, as a SKOS vocabulary and as an RDF document, but they do not provide a SPARQL endpoint. The RDF documents can be downloaded and stored in a local triple store to query the vocabularies using SPARQL.

Cultural Heritage

Europeana[28]

Europeana is a single access point to millions of books, paintings, films, museum objects and archival records that have been digitised throughout Europe. It is an authoritative source of information coming from European cultural and scientific institutions. The content from *Europeana* is available via an API. The *Europeana* LOD pilot, also makes a part of the content available as LOD.

The Data Hub[29]

The Data Hub contains 4.293 datasets that one can browse, learn about and download. In this data hub there are many datasets focusing on art. One can search for the dataset one needs and download it to use as an enrichment source. Examples of interesting datasets for the art sector are:

http://thedatahub.org/group/open-glam

http://thedatahub.org/group/art

[24] http://www.opencyc.org/
[25] http://www.mpi-inf.mpg.de/yago-naga/yago/
[26] http://wordnet.princeton.edu/
[27] http://id.loc.gov/authorities/subjects.html
[28] http://www.europeana.eu
[29] http://thedatahub.org/

http://thedatahub.org/dataset/freebase-visual-art

http://thedatahub.org/dataset/grants-for-the-arts-awards-arts-council-england

http://thedatahub.org/dataset/ukgac

Locations

GeoNames[30]

GeoNames is a geographical database, with information on all countries. It contains over 8 million place names. The data is available as a dump, or via web services.

World Factbook[31]

The CIA World Factbook provides information on the history, people, government, economy, geography, communications, transportation, military and transnational issues of 267 world entities. The World Factbook is available as dump to download.

Persons

VIAF[32]

The Virtual International Authority File initiative is a joint project of several national libraries plus selected regional and trans-national library agencies. The project's goal is to lower the cost and increase the utility of library authority files by matching and linking widely-used authority files and making that information available on the Web. This data source can be used to enrich persons, in particular artists.

Movies

Linked MDB[33]

The Linked Movie DataBase publishes LOD on movies. The Linked MDB also links to DBpedia, YAGO, Flickr wrappr, RDF Book Mashup, MusicBrainz, GeoNames, IMDB, Rotten Tomatoes and Freebase. The content is thus published as LOD and a SPARQL endpoint is made available for querying the dataset.

Music

Music Brainz[34]

MusicBrainz is a music knowledge base. It contains information on artists, release groups, releases, recordings, works, and labels, as well as relationships between them. MusicBrainz data is free to download and is also offered in RDF for download.

BBC Music[35]

BBC Music is also a music knowledge base. It contains information on all the artists ever played at the BBC or in one of their radio shows. It gathers also information from MusicBrainz and Wikipedia. The data is available via LOD, and web services.

[30] http://www.geonames.org/
[31] https://www.cia.gov/library/publications/the-world-factbook/
[32] http://viaf.org/
[33] http://www.linkedmdb.org/
[34] http://musicbrainz.org/
[35] http://www.bbc.co.uk/music

Books

RDF Book Mashup[36]

The RDF Book Mashup makes information available about books, their authors, reviews, and online bookstores. The data is taken from data sources like Amazon, Google, and Yahoo. The information is published on the Web as LOD and the data can be queried using SPARQL.

Proof of concept, developed within the framework of DCA

Our enrichment module will enrich the following entities, as shown in the previous section:

- artwork;
- persons;
- locations;
- artwork type.

For each type, we need to select enrichment sources to search for possible enrichments.

Named Entity Extraction

Before enriching the specified entities, we must first extract the named entities from the EDM instances using OpenCalais. Those to be extracted are: locations, persons, and organisations.

Enrichment Sources Artwork

The artwork, published as LOD, will be linked to its *Europeana* equivalent. Of course, it will need to be published already for *Europeana* to do the enrichment. Another source for artwork enrichment is Freebase, more particular its visual arts collection.

- *Europeana*
- Freebase Visual Art[37]

Enrichment Sources Persons

The actors in the EDM instances will be linked to persons described in DBpedia, Freebase and VIAF. DBpedia and Freebase both serve as enrichment hubs and have extensive information on enrichments for their resources. VIAF, which offers authority records on artists, is used as another enrichment source for persons, in particular artists.

- DBpedia
- Freebase
- VIAF

Enrichment Sources Locations

The locations detected in the EDM instances will be enriched using GeoNames and DBpedia. GeoNames is especially targeted for describing locations. DBpedia is cross-domain, and also has a lot of information on locations.

- GeoNames

[36] http://www4.wiwiss.fu-berlin.de/bizer/bookmashup/
[37] http://www.freebase.com/view/visual_art

- DBpedia

Enrichment Sources Artwork Types

The literals denoting an artwork type will be replaced by a SKOS concept from the DCA vocabulary, discussed in detail in deliverable 3.1 *Metadata implementation guidelines for digitised contemporary artworks* and deliverable 3.2 *Recommendation on contextualisation and interlinking digitised contemporary artworks*.

- DCA SKOS vocabulary

6. LOD strategies for DCA partners

In this section, we provide feasible strategies for DCA partners to publish their content as LOD and to make it possible to interlink all those published entities. The whole workflow has already been discussed and consists of the following steps:

- select appropriate RDF model to publish your data;
- choose an LOD server infrastructure;
- transform your data to RDF;
- enrich your data.

Select appropriate RDF model to publish your data

The DCA partners are institutions on contemporary art. They all hold valuable heritage information. The best model with which to publish this content as LOD is EDM. EDM is geared towards LOD publication of heritage data. At the same time, EDM and LIDO are closely related and the DCA partners have all already mapped their content to LIDO and are therefore familiar with LIDO.

Choose an LOD server infrastructure

The next thing is to choose your LOD infrastructure. For this, the DCA partners have many options, depending on their ICT knowledge and the dynamism of their data.

Organisations with no ICT staff should choose a solution that offers everything to publish the data as LOD. Solutions here are Openlink Virtuoso or D2R Server. Both implement a Linked Data frontend and a SPARQL endpoint over HTTP. Organisations with ICT staff could compile their own solution for LOD publication, just as in our proof of concept. Good triple stores are Openlink Virtuoso and TDB. If you choose Openlink Virtuoso, the Linked Data frontend and SPARQL endpoint are already in place. Otherwise, Pubby is a good solution for making the Linked Data frontend. Joseki or Fuseki are good solutions for providing a SPARQL endpoint over HTTP.

If your data changes a lot and is situated in a relational database, D2R server or Virtuoso Triplify are good solutions. They create an extra layer over your relational database, relying on a JDBC connection to the database. The benefit of this approach is that your data is transformed to RDF on-the-fly. This avoids having syncing problems between the triple store and your relational database. The other benefit of this approach is that the partner's business processes are not affected. They all can just keep on doing their job on the relational database. If you migrate from a relational database to a triple store, these processes will need to be adapted to communicating with it instead of the relational database. From the DCA partner survey it seems that most of them are using a relational database, and that this is the preferred solution. A tutorial on D2R Server will be given to facilitate the process during the last F2F meeting of the DCA partner.

Transform your data to RDF (and ingest)

If you use D2R server or Virtuoso Triplify, you just need to configure the applications to publish your data as RDF. If you migrate all your data to a triple store, you will first need to transform it. This can be done using XSLT or via RDFisers. Many XSLT and RDFisers are available on the Web and free to use for data conversion. Once this is done, the generated RDF data can be ingested into the triple store.

Enrich your data

The only tool used at the moment for enriching data is SILK. A preliminary to using SILK is knowledge of SPARQL. one disadvantage of using SILK is that only data sources can be contacted providing a SPARQL endpoint over HTTP. Another strategy could be to upload your data to *Europeana* and let them enrich it for you. Then *Europeana* can give you back the generated enrichments from *Europeana.* So you could use *Europeana* as an enrichment source.

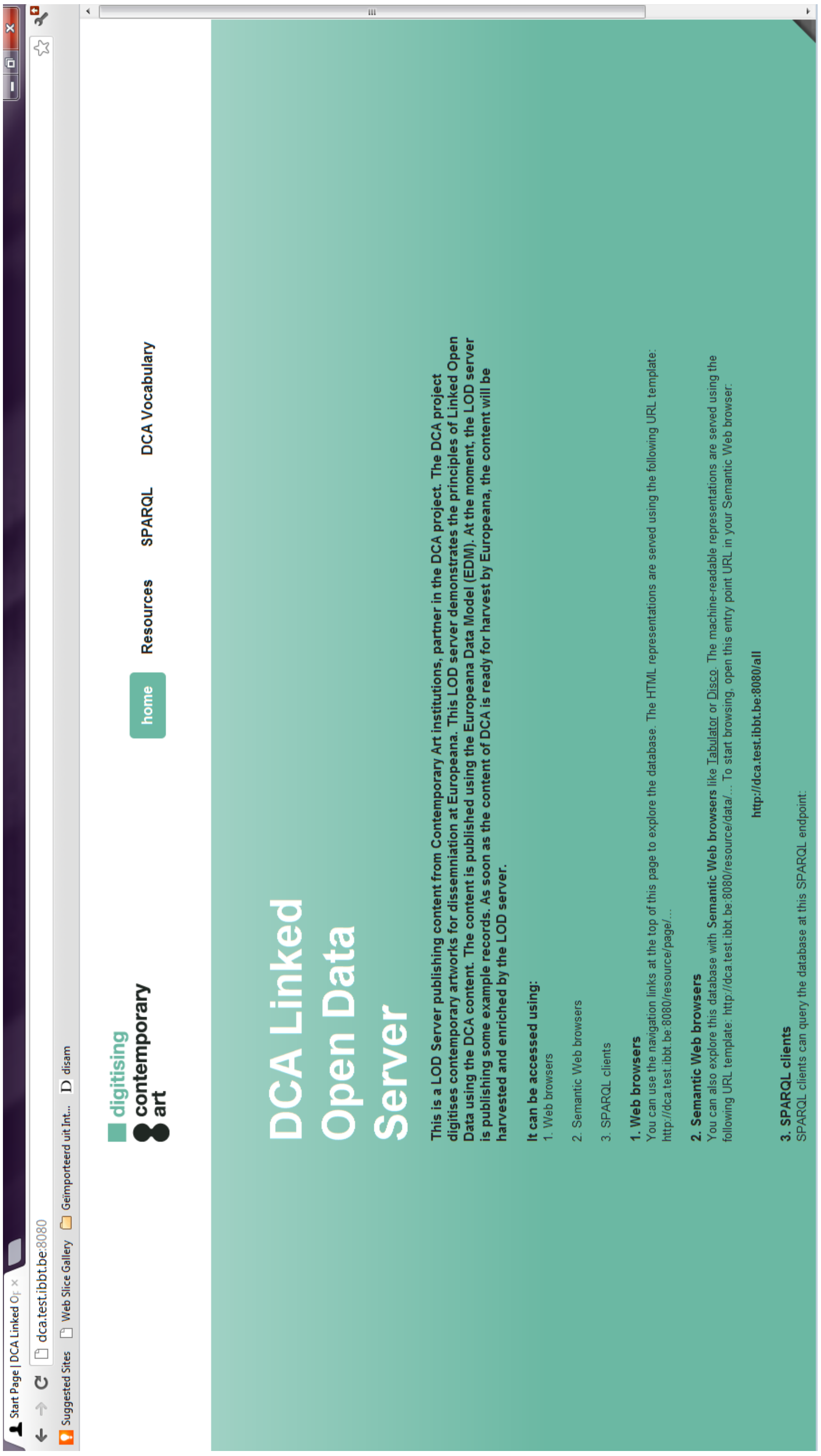
You could also develop your own enrichment module. When doing so, it is good practice to start with OpenCalais to extract the named entities and then use them to look for enrichments in DBpedia. Because DBpedia serves as an enrichment hub, a lot can just be taken directly from it.

7. Bibliography

1. *Linked Data - The Story so far.* Christian Bizer, Tom Heath, and Tim Berners-Lee. 3, s.l. : International Journal on Semantic Web and Information, 2009, Vol. 5.
2. *RDF Primer.* [Online] http://www.w3.org/TR/rdf-primer/.
3. *HTML Specification.* [Online] http://www.w3.org/TR/html401/.
4. *RDF/XML Syntax Specification (Revised).* [Online] http://www.w3.org/TR/rdf-syntax-grammar/.
5. *Turtle - Terse RDF Triple Language.* [Online] http://www.w3.org/TeamSubmission/turtle/.
6. *RDF Vocabulary Description Language 1.0: RDF Schema.* [Online] http://www.w3.org/TR/rdf-schema/.
7. *OWL Web Ontology Language .* [Online] http://www.w3.org/TR/owl-ref/.
8. Dublin Core Metadata Initiative Homepage. [Online] http://dublincore.org/.
9. *Ontology:DOLCE+DnS Ultralite.* s.l. : ontologydesignpatterns.org, 2006.
10. Dan Brickley, Libby Miller. *FOAF Vocabulary Specification 0.98.* 2010.
11. Brickley, Dan. *Basic Geo (WGS84 lat/long) Vocabulary.* s.l. : W3C, 2004.
12. Ryan Shaw, Raphaël Troncy, Lynda Hardman. *LODE: An ontology for Linking Open Descriptions of Events.* 2010.
13. *Europeana Data Model.* [Online] http://group.Europeana.eu/c/document_library/get_file?uuid=718a3828-6468-4e94-a9e7-7945c55eec65&groupId=10605.
14. Dow, Martin. *The OpenART Linked Events Model.* 2010.
15. Hepp, Martin. *GoodRelations Language Reference.* 2011.
16. Carl Lagoze, Herbert Van de Sompel, Michael Nelson, Simeon Warner. *The Open Archives Initiative Protocol for Metadata Harvesting.* 2008.
17. *LIDO - Lightweight Information Describing Objects.* [Online] http://www.lido-schema.org/schema/v1.0/lido-v1.0-specification.pdf.
18. Kay, Michael. *XSL Transformations (XSLT) Version 2.0.* 2007.
19. Antoine Isaac, Ed Summers. *SKOS Simple Knowledge Organization System Primer.* 2009.
20. Ben Adida, Ivan Herman, Manu Sporny, Mark Birbeck. *RDFa 1.1 Primer.* 2012.
21. Eric Prud'hommeaux, Andy Seaborne. *SPARQL Query Language for RDF.* 2007.
22. *SEPIA Data Element Set (SEPIADES).* [Online] http://marinemetadata.org/references/sepiades.
23. *Metadata Encoding and Transmission Standard.* [Online] 2009 йил 29-01. http://www.loc.gov/standards/mets/.
24. *Extensible Markup Language (XML) 1.0 (Fifth Edition).* [Online] http://www.w3.org/TR/REC-xml/.
25. *XML Schema.* [Online] http://www.w3.org/XML/Schema.
26. *Anglo-American Cataloguing Rules.* [Online] http://www.aacr2.org/.
27. *Resource Description and Access.* [Online] http://www.rda-jsc.org/rda.html.
28. International standard bibliographic description (ISBD). [Online] http://archive.ifla.org/VII/s13/pubs/ISBD_consolidated_2007.pdf.

29. *MARC 21 Specifications for Record Structure, Character Sets, and Exchange Media.* [Online] http://www.loc.gov/marc/specifications/.
30. *FUNCTIONAL REQUIREMENTS FOR BIBLIOGRAPHIC RECORDS.* [Online] http://archive.ifla.org/VII/s13/frbr/frbr_current_toc.htm.
31. *Definition of the CIDOC Conceptual Reference Model.* [Online] http://www.cidoc-crm.org/docs/cidoc_crm_version_5.0.3.pdf.
32. *Categories for the Descriptions of Works of Art.* [Online] http://www.getty.edu/research/publications/electronic_publications/cdwa/index.html.
33. *SPECTRUM, the standard for collections management.* [Online] http://www.collectionstrust.org.uk/spectrum.
34. *General International Standard Archival.* [Online] http://www.icacds.org.uk/eng/ISAD(G).pdf.
35. *Encoded Archival Description.* [Online] http://www.loc.gov/ead/.
36. *Art & Architecture Thesaurus.* [Online] http://www.getty.edu/research/tools/vocabularies/aat/about.html.
37. *Getty Thesaurus of Geographic Names.* [Online] http://www.getty.edu/research/tools/vocabularies/tgn/about.html.
38. *Union List of Artist Names.* [Online] http://www.getty.edu/research/tools/vocabularies/ulan/about.html.
39. *Library of Congress Subject Headings.* [Online] http://id.loc.gov/authorities/subjects.html.
40. *RKDartists&.* [Online] http://website.rkd.nl/Databases/RKDartists.
41. *Film identification - Enhancing interoperability of metadata - Element sets and structures.* [Online] http://filmstandards.org/fsc/index.php/EN_15907.
42. *Open Archives Initiative: Object Reuse and Exchange.* [Online] http://www.openarchives.org/ore/1.0/primer.html.
43. *Europeana Semantic Elements Specification.* [Online] http://version1.Europeana.eu/c/document_library/get_file?uuid=77376831-67cf-4cff-a7a2-7718388eec1d&groupId=10128.
44. *Reference Model for an Open Archival Information System.* [Online] http://public.ccsds.org/publications/archive/650x0b1.pdf.
45. *PREMIS Data Dictionary for Preservation Metadata.* [Online] http://www.loc.gov/standards/premis/v2/premis-dd-2-1.pdf.
46. Europeana. *Definition of the Europeana Data.* 2012.

8. Appendix: screenshots of the DCA LOD server



DCA Linked Open Data Server

This is a LOD Server publishing content from Contemporary Art institutions, partner in the DCA project. The DCA project digitises contemporary artworks for dissemniation at Europeana. This LOD server demonstrates the principles of Linked Open Data using the DCA content. The content is published using the Europeana Data Model (EDM). At the moment, the LOD server is publishing some example records. As soon as the content of DCA is ready for harvest by Europeana, the content will be harvested and enriched by the LOD server.

It can be accessed using:
1. Web browsers

2. Semantic Web browsers

3. SPARQL clients

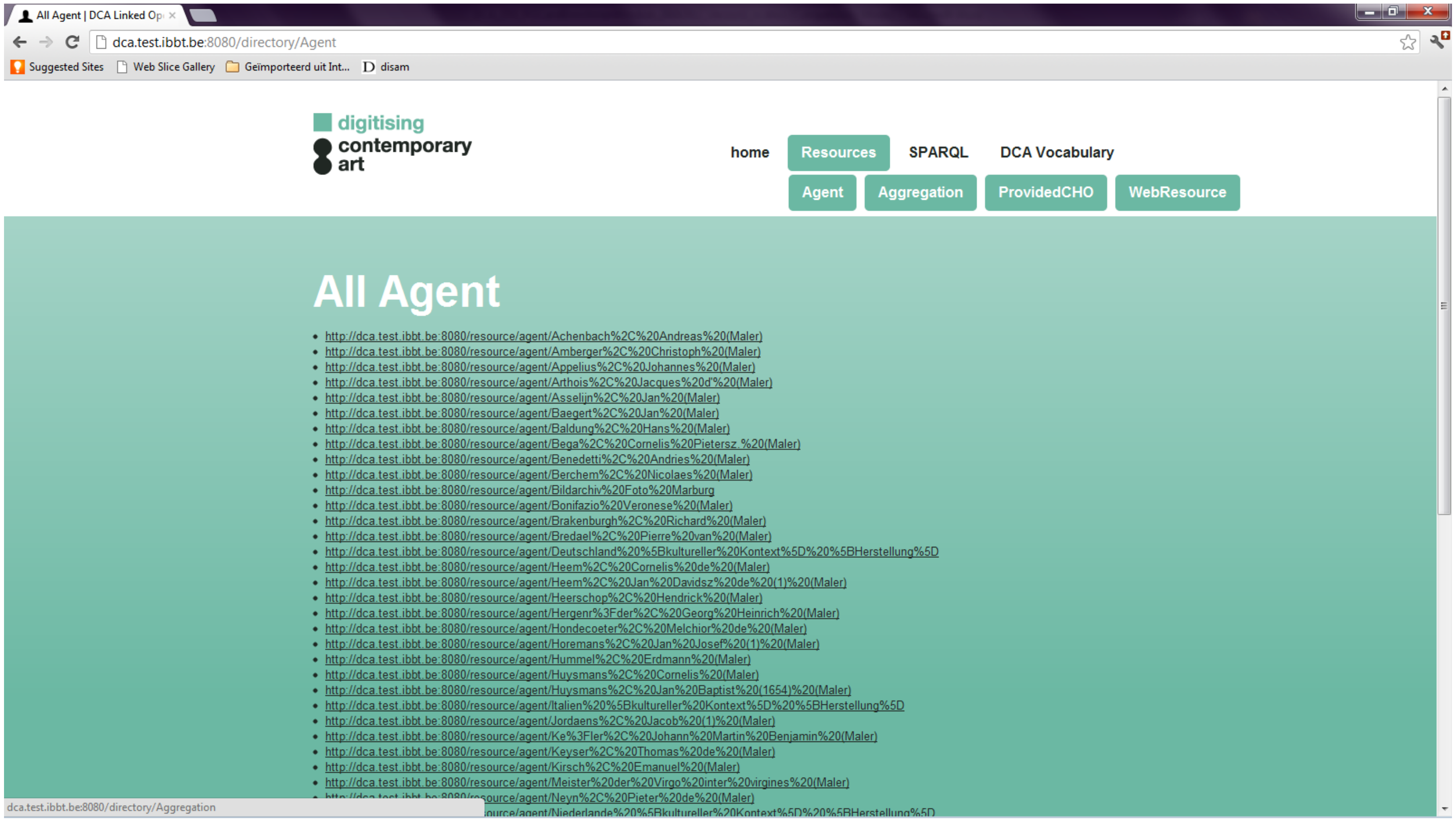
1. Web browsers
You can use the navigation links at the top of this page to explore the database. The HTML representations are served using the following URL template: http://dca.test.ibbt.be:8080/resource/page/...

2. Semantic Web browsers
You can also explore this database with **Semantic Web browsers** like Tabulator or Disco. The machine-readable representations are served using the following URL template: http://dca.test.ibbt.be:8080/resource/data/... To start browsing, open this entry point URL in your Semantic Web browser:

http://dca.test.ibbt.be:8080/all

3. SPARQL clients
SPARQL clients can query the database at this SPARQL endpoint:

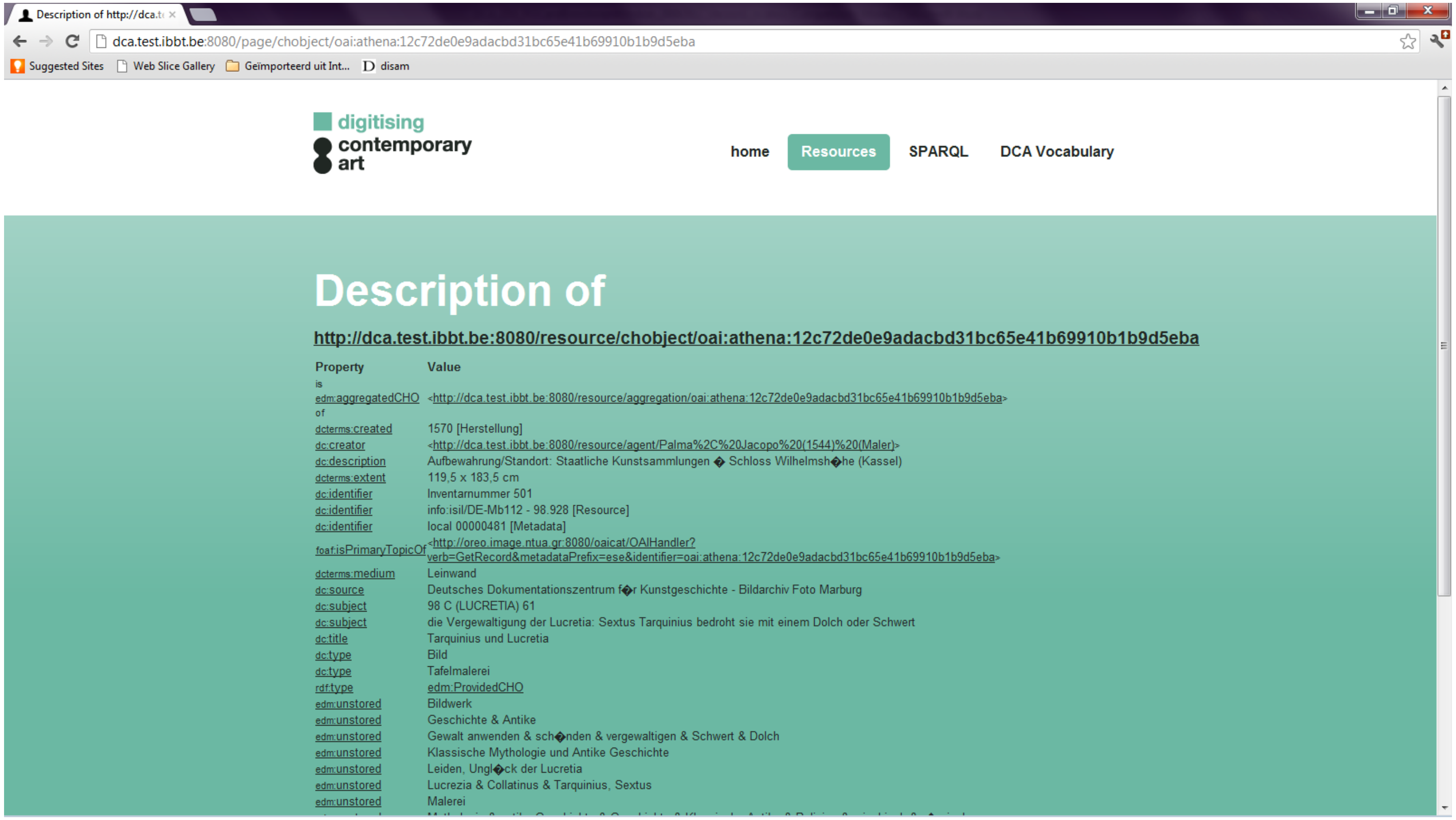
Figure 5: screenshot of the homepage of our PoC


All Agent | DCA Linked Op
dca.test.ibbt.be:8080/directory/Agent
Suggested Sites
Web Slice Gallery
Geïmporteerd uit Int...
disam
digitising contemporary art
home
Resources
SPARQL
DCA Vocabulary
Agent
Aggregation
ProvidedCHO
WebResource
dca.test.ibbt.be:8080/directory/Aggregation

All Agent

- http://dca.test.ibbt.be:8080/resource/agent/Achenbach%2C%20Andreas%20(Maler)
- http://dca.test.ibbt.be:8080/resource/agent/Amberger%2C%20Christoph%20(Maler)
- http://dca.test.ibbt.be:8080/resource/agent/Appelius%2C%20Johannes%20(Maler)
- http://dca.test.ibbt.be:8080/resource/agent/Arthois%2C%20Jacques%20d'%20(Maler)
- http://dca.test.ibbt.be:8080/resource/agent/Asselijn%2C%20Jan%20(Maler)
- http://dca.test.ibbt.be:8080/resource/agent/Baegert%2C%20Jan%20(Maler)
- http://dca.test.ibbt.be:8080/resource/agent/Baldung%2C%20Hans%20(Maler)
- http://dca.test.ibbt.be:8080/resource/agent/Bega%2C%20Cornelis%20Pietersz.%20(Maler)
- http://dca.test.ibbt.be:8080/resource/agent/Benedetti%2C%20Andries%20(Maler)
- http://dca.test.ibbt.be:8080/resource/agent/Berchem%2C%20Nicolaes%20(Maler)
- http://dca.test.ibbt.be:8080/resource/agent/Bildarchiv%20Foto%20Marburg
- http://dca.test.ibbt.be:8080/resource/agent/Bonifazio%20Veronese%20(Maler)
- http://dca.test.ibbt.be:8080/resource/agent/Brakenburgh%2C%20Richard%20(Maler)
- http://dca.test.ibbt.be:8080/resource/agent/Bredael%2C%20Pierre%20van%20(Maler)
- http://dca.test.ibbt.be:8080/resource/agent/Deutschland%20%5Bkultureller%20Kontext%5D%20%5BHerstellung%5D
- http://dca.test.ibbt.be:8080/resource/agent/Heem%2C%20Cornelis%20de%20(Maler)
- http://dca.test.ibbt.be:8080/resource/agent/Heem%2C%20Jan%20Davidsz%20de%20(1)%20(Maler)
- http://dca.test.ibbt.be:8080/resource/agent/Heerschop%2C%20Hendrick%20(Maler)
- http://dca.test.ibbt.be:8080/resource/agent/Hergenr%3Fder%2C%20Georg%20Heinrich%20(Maler)
- http://dca.test.ibbt.be:8080/resource/agent/Hondecoeter%2C%20Melchior%20de%20(Maler)
- http://dca.test.ibbt.be:8080/resource/agent/Horemans%2C%20Jan%20Josef%20(1)%20(Maler)
- http://dca.test.ibbt.be:8080/resource/agent/Hummel%2C%20Erdmann%20(Maler)
- http://dca.test.ibbt.be:8080/resource/agent/Huysmans%2C%20Cornelis%20(Maler)
- http://dca.test.ibbt.be:8080/resource/agent/Huysmans%2C%20Jan%20Baptist%20(1654)%20(Maler)
- http://dca.test.ibbt.be:8080/resource/agent/Italien%20%5Bkultureller%20Kontext%5D%20%5BHerstellung%5D
- http://dca.test.ibbt.be:8080/resource/agent/Jordaens%2C%20Jacob%20(1)%20(Maler)
- http://dca.test.ibbt.be:8080/resource/agent/Ke%3Fler%2C%20Johann%20Martin%20Benjamin%20(Maler)
- http://dca.test.ibbt.be:8080/resource/agent/Keyser%2C%20Thomas%20de%20(Maler)
- http://dca.test.ibbt.be:8080/resource/agent/Kirsch%2C%20Emanuel%20(Maler)
- http://dca.test.ibbt.be:8080/resource/agent/Meister%20der%20Virgo%20inter%20virgines%20(Maler)
- http://dca.test.ibbt.be:8080/resource/agent/Neyn%2C%20Pieter%20de%20(Maler)
- source/agent/Niederlande%20%5Bkultureller%20Kontext%5D%20%5BHerstellung%5D

Figure 6: screenshot of webpage listing all edm:Agent entities of the PoC

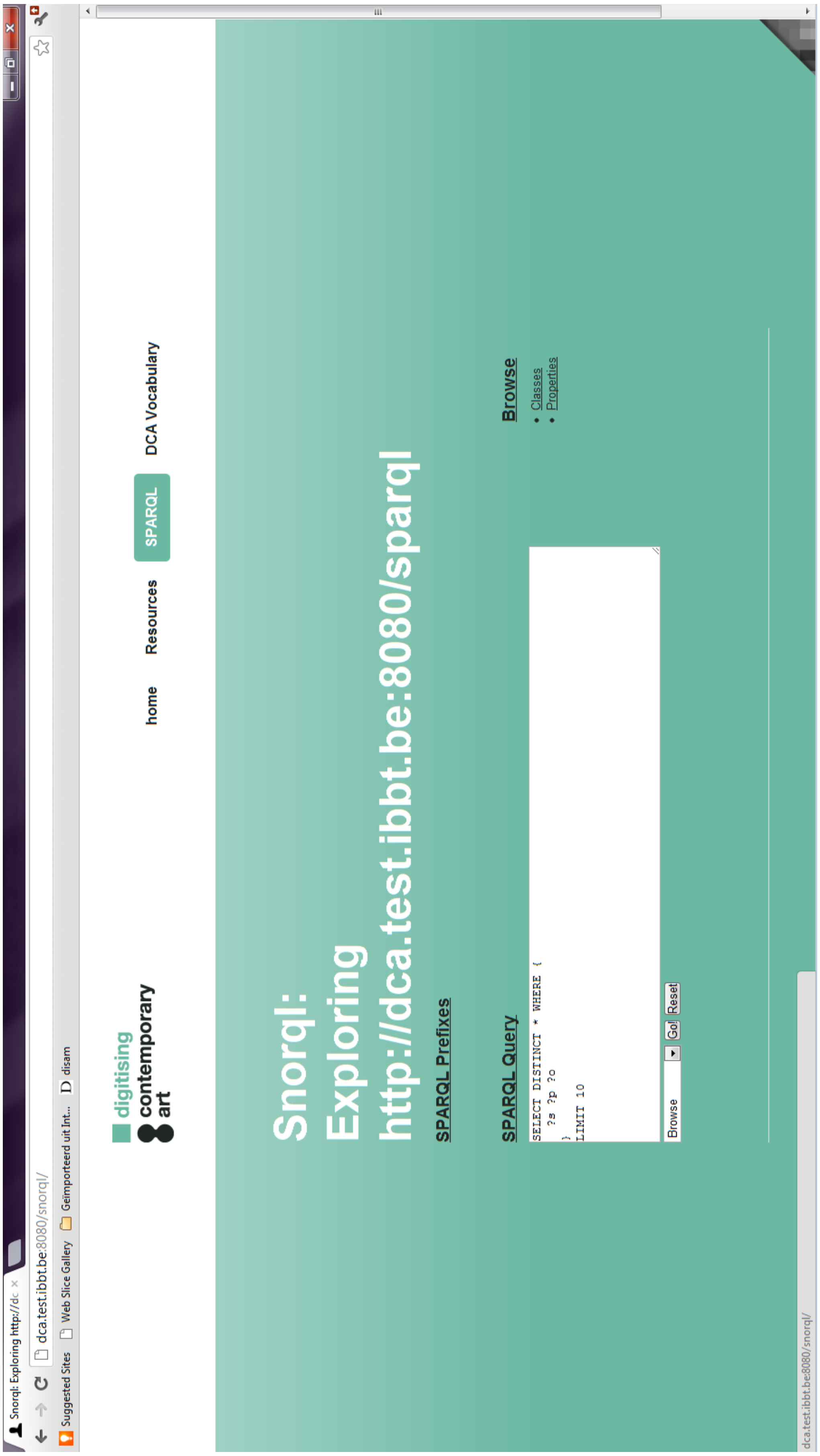

Description of http://dca.t
dca.test.ibbt.be:8080/page/chobject/oai:athena:12c72de0e9adacbd31bc65e41b69910b1b9d5eba
Suggested Sites
Web Slice Gallery
Geïmporteerd uit Int...
disam
digitising contemporary art
home
Resources
SPARQL
DCA Vocabulary

Description of

http://dca.test.ibbt.be:8080/resource/chobject/oai:athena:12c72de0e9adacbd31bc65e41b69910b1b9d5eba

Property	Value
is edm:aggregatedCHO of	<http://dca.test.ibbt.be:8080/resource/aggregation/oai:athena:12c72de0e9adacbd31bc65e41b69910b1b9d5eba>
dcterms:created	1570 [Herstellung]
dc:creator	<http://dca.test.ibbt.be:8080/resource/agent/Palma%2C%20Jacopo%20(1544)%20(Maler)>
dc:description	Aufbewahrung/Standort: Staatliche Kunstsammlungen � Schloss Wilhelmsh�he (Kassel)
dcterms:extent	119,5 x 183,5 cm
dc:identifier	Inventarnummer 501
dc:identifier	info:isil/DE-Mb112 - 98.928 [Resource]
dc:identifier	local 00000481 [Metadata]
foaf:isPrimaryTopicOf	<http://oreo.image.ntua.gr:8080/oaicat/OAIHandler?verb=GetRecord&metadataPrefix=ese&identifier=oai:athena:12c72de0e9adacbd31bc65e41b69910b1b9d5eba>
dcterms:medium	Leinwand
dc:source	Deutsches Dokumentationszentrum f�r Kunstgeschichte - Bildarchiv Foto Marburg
dc:subject	98 C (LUCRETIA) 61
dc:subject	die Vergewaltigung der Lucretia: Sextus Tarquinius bedroht sie mit einem Dolch oder Schwert
dc:title	Tarquinius und Lucretia
dc:type	Bild
dc:type	Tafelmalerei
rdf:type	edm:ProvidedCHO
edm:unstored	Bildwerk
edm:unstored	Geschichte & Antike
edm:unstored	Gewalt anwenden & sch�nden & vergewaltigen & Schwert & Dolch
edm:unstored	Klassische Mythologie und Antike Geschichte
edm:unstored	Leiden, Ungl�ck der Lucretia
edm:unstored	Lucrezia & Collatinus & Tarquinius, Sextus
edm:unstored	Malerei

Figure 7: screenshot showing the description of an edm:ProvidedCHO instance of the PoC


Snorql: Exploring http://dca.test.ibbt.be:8080/sparql

home Resources SPARQL DCA Vocabulary

SPARQL Prefixes

SPARQL Query

```
SELECT DISTINCT * WHERE {
  ?s ?p ?o
}
LIMIT 10
```

Browse Go! Reset

Browse
- Classes
- Properties

Figure 8: screenshot of the SPARQL endpoint interface of PoC