Bringing DSpace into the Semantic Web

Outdated

The information here is slightly outdated. An up to date documentation can be found in the official DSpace Documentation, starting with DSpace 5: Linked (Open) Data support of DSpace 5.

Repositories and the Semantic Web

The most sites on the Internet are oriented towards human consumption. While HTML may be a good format to create websites it is not a good format to export data in a way a computer can work with. Like the most software for repositories DSpace supports OAI-PMH as an interface to export the stored data. While OAI-PMH is well known in the field of repositories it is rarely known elsewhere (e.g. Google retired its support for OAI-PMH in 2008). The Semantic Web is an approach to publish data on the Internet together with information about its semantics. The W3C released standards like RDF or SPARQL for publishing structured data on the Web in a way computers can easily work with. The data stored in repositories is particularly suited to be used in the Semantic Web, as metadata is already available. It doesn't have to be generated or entered manually for publication as Linked Data. For most repositories, at least for Open Access repositories, it is quite important to share their stored content. Linked Data is a rather big chance for repositories to present their content in a way it can easily be accessed, interlinked and (re)used.

To my knowledge, EPrints is currently the only repository software capable to export its content as RDF. Nevertheless, the software ignores some important conventions regarding Linked Data, meaning it rather provides RDF than Linked Data.

The main topics of my thesis were how metadata and digital objects stored in repositories can be woven into the Linked (Open) Data Cloud and which characteristics of repositories have to be considered while doing so. As main part of my thesis I created a software independent concept on how to provide repository contents as Linked Data. In addition, I implemented it as a DSpace extension. There are some last steps left to be done before it can be used in a productive environment. I would be glad to contribute it to a future release of DSpace as soon as it's ready.

German native speakers can find my thesis here: http://www.pnjb.de/uni/diplomarbeit/repositorien_und_das_semantic_web.pdf.

The JIRA-Ticket for this contribution can be found here: ~~DS-2061~~ - Linked (Open) Data support for DSpace CLOSED .

I created a pull request on github, to share the code as early as possible: https://github.com/DSpace/DSpace/pull/568.

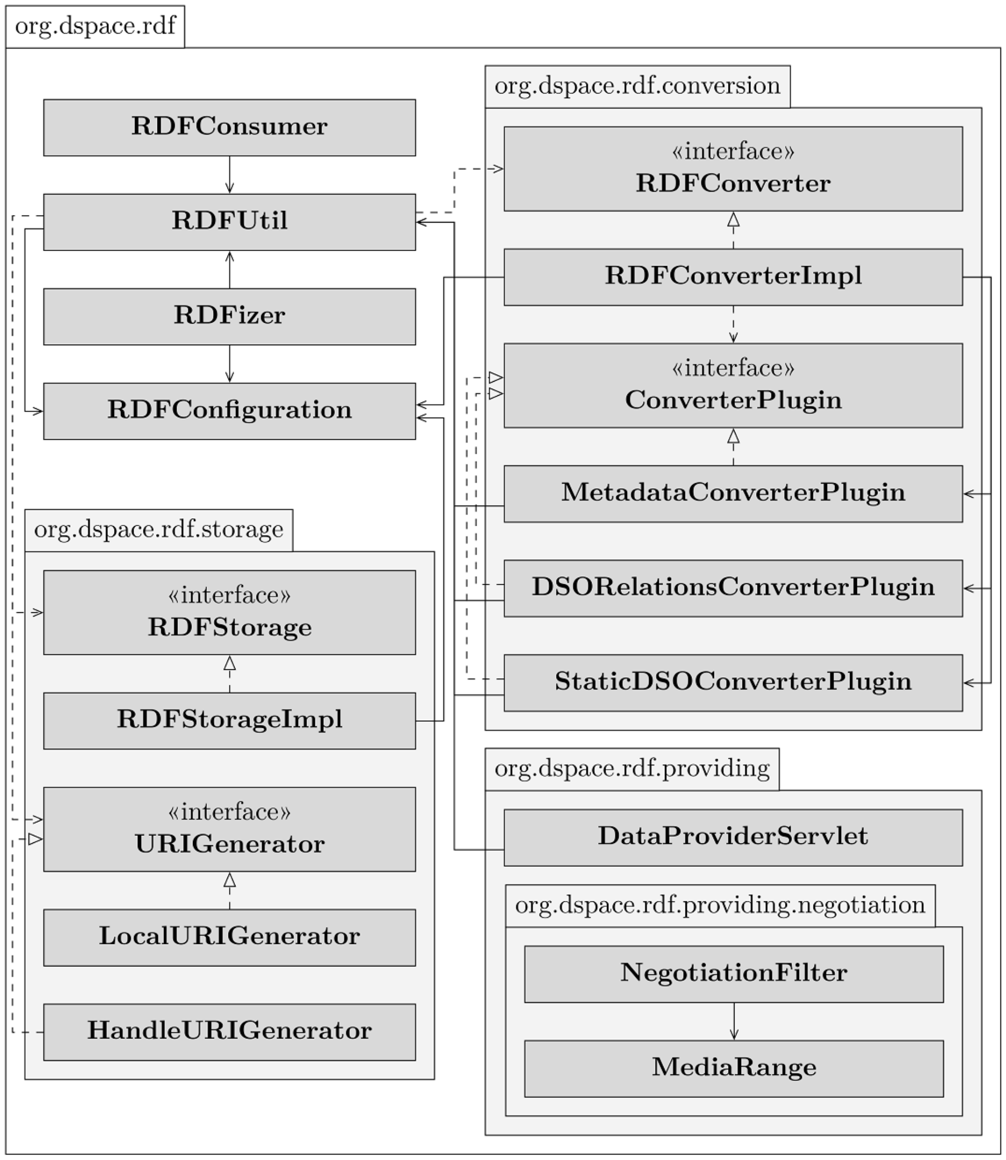
State of this document

This document is a rather small documentation of dspace-rdf, but I wanted to share this contribution as early as possible. As I developed dspace-rdf as part of my thesis I was not allowed to speak about it before I handed my thesis in to my university. So this document should be the basis to introduce dspace-rdf and to discuss design decisions I have made. German native speakers should take a look inside my thesis. Chapters 4.2 and 5.1 explain the ideas behind the concept, while chapters 4.3 and 5.2 documents the implementation. Chapter 5.3 contains the things to be done before dspace-rdf can be used in a productive environment. On topic left to be done actually is a good english documentation.

dspace-rdf

dspace-rdf is an extension for DSpace that adds capabilities to convert contents stored in DSpace into RDF, to store the converted data in a Triple Store and to provide it as serializations of RDF. The Triple Store must support SPARQL 1.1 and can be used to provide the converted data over a read-only SPARQL endpoint. dspace-rdf can currently be found on my github repository, but I would be glad to contribute it to a future version of DSpace.

dspace-rdf is realized as a new module of DSpace as it contains a webapp and everyone should be able to decide whether it should be deployed or not. dspace-rdf contains several parts. You can see a simplified class diagramm here:



The classes RDFConsumer and RDFUtil integrate dspace-rdf into DSpace. RDFConfiguration is used to centralize configuration properties used by more than one class inside dspace-rdf. The class RDFizer contains the command line interface. To see the online help you have to use the following command:

```
[dspace-install]/bin/dspace dsrun org.dspace.rdf.RDFizer --help
```

The online help should give you all necessary information.

The package org.dspace.rdf.storage contains the classes to attach the Triple Store to the repository. The URIGenerator is an interface used to create URIs that are used to identify the repository resources in RDF. In the configuration (see below) you can define which class should be used. In cause of DS-1990 (~~DS-1990~~ - Missing a way to handle identifiers like DOIs in case of a deletion event CLOSED) you currently have the choice to use local URIs or Handles in the form of http URIs (g.e. http://hdl.handle.net/123456789/1). As soon as DS-1990 is resolved at least a DOIURIGenerator should be added.

The package org.dspace.rdf.conversion contains the classes used to convert the repository's content to RDF. The conversion itself is done by plugins. The Interface org.dspace.rdf.conversion.ConverterPlugin is really simple, so take a look if you want to extend the conversion. The only thing important is, that the plugins must only create RDF that can be made publicly available (see Design Decisions for further information). The MetadataConverterPlugins is heavily configurable (see below) and is used to convert metadata of Items. The StaticDSOConverterPlugin can be used to add static RDF Triple (see below). The DSORelationsConverterPlugin is not configurable yet. It is just a proof of concept and its revision is on the TODO list below.

The package org.dspace.rdf.providing contains the servlet to provide the converted data as serialized RDF. In the last days I added a second servlet, which is not part of the image yet. The LocalURIRedirectionService is used if you use the LocalURIGenerator. Linked Data distinguishes between non-information and information-resources. If a URI of a non-information resource is dereferenced it should forward to the URL of an information resource describing the non-information resource. With Handles or DOIs the resolver will perform the forward. For local URIs the LocalURIRedirectionService will do. The package org.dspace.rdf.providing.negotiation contains the classes for content negotiation. Linked Data uses content negotiation to serve the same information as html or different serializations of RDF under the same URI.

Installation

The installation follows the normal installation of a DSpace source release. In addition you have to provide a Triple Store. You can use any Triple Store you like, if it support SPARQL 1.1 Query Language and SPARQL 1.1 Graph Store HTTP Protocol. If you do not have one yet, you can use Apache Fuseki. Download Fuseki from its official download page and unpack the downloaded archive. The archive contains several scripts to start fuseki. Use the start script appropriated for the OS of your choice with the options '--localhost --config=<dspace-install>/config/modules/rdf/fuseki-assembler.ttl'. Instead of changing into the directory you unpacked fuseki to, you may set the variable FUSEKI_HOME. If you're using Linux, unpacked fuseki to /usr/local/jena-fuseki-1.0.1 and installed DSpace to /var/dspace this would look like this:

```
 export FUSEKI_HOME=/usr/local/jena-fuseki-1.0.1 ; $FUSKI_HOME/fuseki-server --localhost --config /var/dspace
/config/modules/rdf/fuseki-assembler.ttl
```

Fuseki's archive contains a script to start fuseki automatically at startup as well.

The configuration of Fuseki DSpace provides configures it to provide several SPARQL endpoints, even some that can be used to change the data of the Triple Store. You should not use this configuration and let Fuseki connect to the internet as it would make it possible for anyone to delete, change or add information to the Triple Store. The option --localhost tells fuseki to listen only on the loopback device. You can use Apache mod_proxy to make the read-only SPARQL endpoint accessible from the internet. A more detailed documentation on how this can be done will follow here one day.

Configuration

The file [dspace-install]/config/modules/rdf.cfg is the main configuration file of dspace-rdf. It contains information on how to connect to the Triple Store, which class should be used as URIGenerator, which plugins will convert the repository content to RDF, which URL is used in the content negotiation to reach the DataProviderServlet and so on. It contains a lot of comments, so please take a look into the configuration file until there is a better documentation of dspace-rdf.

In the folder [dspace-install]/config/modules/rdf/ you'll find more configuration files. The configuration files are written in RDF or more specific in Turtle. This behavior makes it easy to specify triples that should be part of the converted data and has some other advantages as well.

The files named [dspace-install]/config/modules/rdf/contstant-data-*.ttl are used to configure the StaticDSOConverterPlugin. Every triple specified in the file constant-data-general.ttl will be added to every resource created while conversion. For each DSpaceObject type except for Bundles and Bitstreams one file exists, that can be used to specify RDF that will be added to the converted data representing a resource of the appropriate DSpaceObject type. Bundles and Bitstreams gets converted as part of the Item they belong to.

The files [dspace-install]/config/modules/rdf/metadata-*.ttl configure the MetadataConverterPlugin. This is the plugin that converts Item's metadata. The file metadata-prefixes.ttl can be used to specify prefixes (or namespaces) to be used in those serializations of RDF that support such mechanism (g.e. Turtle or RDF/XML). The file metadata-rdf-schema.ttl is not a configuration file. It contains the schema describing the vocabulary used for the configuration of the MetadataConverterPlugin. The file metadata-rdf-mapping.ttl contains several mappings between metadata fields and triples that should be created to represent the specified metadata field. It already contains some examples which should help until it is better documented. The triples that should be created while converting a metada field are specified using RDF reification. You can specify regular expresions to create some triple only for metadata fields those values fulfils the specified regular expression. Two more regular expressions can be used to manipulate field values before they will be used to generate Literals or URIs. The file metadata-rdf-mapping.ttl already contains examples on how to generate URIs and Literals using metadata field values. When the metadata-rdf-mapping.ttl is loaded simple inferencing is used to detect most of the types of its entities. So you do not need to type every resource. Typing (by using statements with the property 'rdf:type' or the turtle shortcut 'a') is necessary for ResourceGenerators and LiteralGenerators only as there is no way to distinguish them using inferencing. While links are what distinguish Linked Data from simple RDF, it is important to use ResourceGenerators and regular expresions to create links.

RDFizer

The RDFizer is a command line interface administrators can use to convert the complete repository contents, some content specified by its handle or to delete data from the Triple Store. If the Triple Store is reachable the RDFConsumer converts data at the moment it is changed within DSpace so that the Triple Store should stay synchronized with the repository. You can get the online help by executing the following command:

```
[dspace-install]/bin/dspace dsrun org.dspace.rdf.RDFizer --help
```

If you add dspace-rdf to an existing DSpace instance you should run the RDFizer at leas once to initially convert already existing Items, Collections and Communities.

Development / API

If you want to use RDF in other parts of DSpace you should take a look on the class org.dspace.rdf.RDFUtil as it acts as interface between dspace-rdf and the other parts of DSpace.

Beside RDFUtil you can use the classes in the package org.dspace.rdf.providing.negotiation for content negotiation. I added content negotiation to JSPUI already, but it should be added to XMLUI as well. It would be even better to use content negotiation directly in Persistent Identifier Resolvers that support content negotiation. The resolver at dx.doi.org supports content negotiation, but the DOIIdentifierProviders (currently contains a DOIIdentifierprovider for EZID and on for DataCite) must be extended before the content negotiation can be used.

TODOs

There are some things left to be done before dspace-rdf should be used in a productive environment. The most important topic is a good default configuration as it is configurable which metadata fields gets converted, which vocabularies gets used and which links are generated. While I added some default configuration already, it should be discussed which vocabularies should be used, which links should be generated and so on. G.e. it is impossible to automatically convert the most Bitstreams. But they should be linked at least. I'm not sure which vocabulary should be used to link them. While EPrints made some design decisions I don't follow, they already developed a vocabulary to describe repositories in RDF. We should take a look at the EPrints Ontology and decide whether it can be used in DSpace, whether it must be extended or whether a more generic Ontology could be developed to describe repositories independent from the software that is used to realize them.

~~The DSORelationsConverterPlugin is not configurable yet. It was just a proof of concept on how to interlink communities, collections, items and bitstreams.~~ This is done with the commit 526a364 (updated 2014-09-01).

~~As soon as DS-1990 (~~ ~~DS-1990~~ - Missing a way to handle identifiers like DOIs in case of a deletion event CLOSED ~~) is resolved the interface URIGenerator should be changed to use the Identifiers array the PR for DS-1990 suggest. Then a DOIURIGenerator should be developed so that DOIs in the form http://dx.doi.org/<doi> can be used to identify resources in RDF. The DOIIdentiferProviders (the one for DataCite and the other one for EZID) could be enhanced to activate content negotiation.~~ I have a commit ready for this, but I'm waiting for my PR #537 to be merged and to solve DS-1990 (updated 2014-09-01).

The XMLUI and the JSPUI should use link-tags linking to the serializations of the converted RDF as in the following example:

```
<link rel="alternate" type="application/rdf+xml" href="http://demo.dspace.org/rdf/handle/123456789/123/rdf" />
<link rel="alternate" type="text/turtle" href="http://demo.dspace.org/rdf/handle/123456789/123/ttl" />
```

And of course a good English documentation is necessary as this document only gives a little peak on how to configure and use dspace-rdf.

Design Decisions

I decided to add a Triple Store to the repository so that no data needs to be converted at the moment the converted data is accessed. This decision was done with the idea in mind that contents of repositories will much more often be read as changed. To avoid big changes in the core of DSpace and to make it easy to use dspace-rdf in existing repositories I decided that the Triple Store should extend the repository and not replace the relational database. The Triple Store can be considered as a cache for the converted data. Beside that it should be used to provide a read-only worldwide accessible SPARQL endpoint containing all converted data. The Triple Store should contain only data that is public as the access restriction of DSpace won't affect the SPARQL endpoint. For this reason dspace-rdf converts only archived, discoverable (non-privat) Items, Collections and Communities that are readable for anonymous users. Plugins converting Item metadata should check whether as specific metadata field needs to be protected or not (see org.dspace.app.util.MetadataExposure on how to check that).

Work in progress

Work in progress

The following information will become part of this documentation as soon as they are ready. The following part of this document is work in progress and not finished yet.

DSpace comes with Dublin Core as default metadata schema and contains several metadata fields out of the box. While new metadata schemas and fields can be added or existing once can be deleted many DSpace users will use the default metadata fields and extend them if necessary. This in mind we should create a default configuration to convert the default metadata fields of DSpace into RDF. This configuration can easily be adjusted to the special needs of a repository or extended to export local fields. Beside the metadata we should describe the repository itself in RDF and convert the structure of the repository (communities and collections) as well. The following part documents the default configuration of the rdf module.

Metadata Conversion

Describing the repository and its structure in RDF

To describe the repository and its structure (communities and collections) a DSpace Repository Ontology was created. The DSpace Repository Ontology contains classes like Repository, Community, Collection and Item. The class Repository is a subclass of void:Dataset. Everyone using dspace-rdf should consider to add a description of the repository as a Dataset. This can be done with the StaticDSOConverterPlugin as described above. The following part is an example for the contents of the file [dspace-install]/config/modules/rdf/contstant-data-site.ttl which can be used to add static to the description of the repository itself:

[dspace-install]/config/modules/rdf/constant-data-site.ttl

```
@prefix void: <http://rdfs.org/ns/void#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .

@prefix : <#> .

# void expects the following URI to be a unique reference to the description of an single repository only.
# See http://www.w3.org/TR/2011/NOTE-void-20110303/#webpage for further information.
<>  foaf:homepage <http://dspace.demo.org/jspui> ;
# you may add a link to the homepage of the organization running the repository this way.
    foaf:page <http://www.dspace.org> ;
# you can use dublin core terms to describe the repository, see http://www.w3.org/TR/2011/NOTE-void-20110303
/#dublin-core
    dcterms:title "DSpace Demo Repository" ;
    dcterms:publisher :DSpaceCommiters ;
    dcterms:description "This is a demonstration instance of DSpace. This repository is not a repository that
gains persistence or usable content. It is a sandbox repository demonstrating a repository sofware." ;
# use some more information described by void (see http://www.w3.org/TR/2011/NOTE-void-20110303/).
    void:sparqlEndpoint <http://demo.dspace.org/tripestore/dspace/sparql> ;
    void:feature <http://www.w3.org/ns/formats/N3> ;
    void:feature <http://www.w3.org/ns/formats/N-Triples> ;
    void:feature <http://www.w3.org/ns/formats/RDF_XML> ;
    void:feature <http://www.w3.org/ns/formats/Turtle> ;
    void:rootResource <> ;
    .

# Describe the publisher:
:DSpaceCommiters    a               foaf:Organization ;
                    foaf:homepage   <https://wiki.duraspace.org/display/DSPACE/DSpaceContributors> ;
                    .
```