The DITA RDF project

Contents

Welcome to the DITA RDF project

The objective of this project is to develop an ontology to describe DITA XML objects and to develop tools to generate RDF triples based on that ontology.

The source code is published *on Github*.

Objective

This poster illustrates the objective of the project. I presented it during the *10th Summer School on Ontology Engineering and the Semantic Web* (SSSW 2013).

This project aims at providing the necessary tools to extract and publish the metadata of the documentation. Then, this metadata can be linked with other data types (product, people, sales, non-DITA document metadata, etc.) to create a powerful organization graph.

Although this project only covers the extraction and publication of the DITA metadata, I can help you with *the next steps*.

Status

Table 1: Summary of the deliverables

Deliverable	Status	Description
Ontology	80% (*see issues in Github*)	The DITA RDF ontology expresses the semantics of the DITA specification in RDF.
DITA OT plugin	90% of the ontology is supported (*see issues in Github*)	A DITA Open Toolkit plugin to parse DITA content and extract the valuable metadata in RDF/XML.
SPARQL queries	Not started	A set of SPARQL queries to get you started with querying the extracted metadata graph.
Dashboarding tools	Not started	*Sgvizler* and *Sparklis* are the candidate tools.
Componize pipeline plugin	Not started (*see issues in Github*)	A plugin that would enable Componize users to run the dita2rdf transformation.

Project background

This idea came to me when we (*NXP Customer Documentation Services*) *decided* to publish the metadata of our products and documents as RDF in order to solve the siloization of our various data types, define company-wide data models and so that external organizations can easily consume this data.

I set up the publication of the metadata about binary documents (data sheets, user manuals, etc.), but since our DITA content was built on complex relationships (topic reuse) and was our primary format for translation, we needed a proper ontology to describe it. The driver was to provide detailed metrics to authors and greatly facilitate certain internal processes such as the selection of the content to translate or the detection of inconsistencies in the status of documents. Indeed, we considered that being able to formulate multidimensional queries across topics, maps, users, products and company divisions was a major step ahead.

The driver was consequently mostly internal, but we plan to publish a subset of this metadata when we go live with the whole public data set. Follow *@nxpdata* to get the latest updates about the linked data published by NXP.

Finally, since the challenges we faced were most likely similar to the ones faced by other DITA-enabled organizations, I decided to share my findings.

John Walker and Tim Nelissen, from NXP CDS, have since then founded *Semaku*, a consultancy and software company dedicated to helping organizations to solve their data and content silo issues using the *linked data* principles. I joined them in April 2014.

They talk about the DITA RDF project

Links to pages or videos in which the DITA RDF project is mentioned

Date	Description	Link
August 12th, 2014	*Gary Russo*, from *Harper Collins*, describing how he could use the dita2rdf DITA OT plugin to export the metadata of the DITA content in *RSuite CMS*, based on a *MarkLogic database*	*Presentation, 1.5 MB*
November 19th, 2013	Myself, presenting the DITA RDF project at *DITA Europe 2013*.	Streamed video, 30 minutes

RDF for the beginner

It's just another way to express data.

First things first, the *Resource Description Framework (RDF)*, is a W3C recommendation since 2003, with an update to version 1.1 in 2014. Its purpose is to achieve what the Web has enabled for documents, but this time with data. Since the dawn of the Web, one of its main strength is to enable users to interlink documents easily, building browsing paths for the Web users. One Web page can be the starting point to an infinity of other Web pages. With RDF, it's the same, but with data.

RDF 101

A great introduction to RDF for the beginners

RDF 101 by Cambridge Semantics

Various data models compared with RDF (written for beginners!)

To get a better understanding of what would data interlinking mean, let's have a look at how data is usually managed today. I am not a data scientist, thus I will stick to the most traditional principles.

In order to compare these data models, we will use the same set of data and represent it in each model. Here is the data, in natural language:

> Eric Landais, 35 years old, and Olga Landais, 36 years old, have two children, Salomé and Isaac.
>
> Salomé is 9 years old and Isaac is 5 years old.
>
> Olga has a brother, Boris Todorov, 42 years old.
>
> Olga and Eric go together to an event.
>
> Olga and Boris go together to another event.
>
> Salomé goes to yet another event.

The relational database table

Relational databases are useful for tabular data, but limited due to the lack of standardization

Relational database management systems (RDBMS) are the most common way to store data. The most famous solutions are MySQL and Oracle. In these data bases, the data is stored in tables. Each table can have virtually infinite number of columns and rows, and each data entry is stored as a new row. Here is a typical example:

id	firstName	familyName	age	parent	sibling	child	goesToEvent
1	Salomé	Landais	9	3,4	2		11
2	Isaac	Landais	5	3,4	1		
3	Eric	Landais	34			1,2	5
4	Olga	Landais	35			1,2	5,8
5	Boris	Todorov	42	9	4		8
...	...	...	...	...	...	...	...

This table stores the name and the age of people, including how they are related. Each person has a unique identifier in the scope of this table, which enables the creation of relationships across the entries of the table. It is agreed that the last column refers to the rows of another table that stores events, each event also having a unique identifier in the scope of the events table. The identifiers to data entries in another table are called foreign keys.

When it comes to data interlinking, we meet certain problems that are inherent to the way relational databases are meant to work.

Contrary to the Web of documents that uses standards to present the information (HTML, W3C standard) and transport it (HTTP, IETF standard), relational databases are not supported by standard vocabularies or models.

If company Y wants to interchange data with company X, even if they use the same database vendor, they cannot interchange data easily, because they probably use different column headers and made different choices regarding how the data will be distributed across the tables.

Finally, unique identifiers are only unique in a limited scope. In worst cases they are unique in the scope of an individual table, in best cases they are unique in a circle of organizations that agreed on an identification system. This means that when the data is merged with another data set, there is a risk to end up with data entries that have the same identifier, which results in data clashes. To get a feeling of this situation, imagine two cars in the same country having the same license plate, or two phone chips being bound to the same phone number.

On the Web of documents, we have robust worldwide identifiers: Universal Resource Locators (URL). If you visit *https://github.com/ColinMaudry/dita-rdf*, thanks to standards, you are certain of a couple things:

- You can access this content from any computer on the planet that has Internet access, whatever operating system or Internet browser you use
- The content published at this address is, in a certain extent, under the responsability of Github
- The transmission of information between your computer and the server is encrypted (HTTPS)

The XML tree

XML is versatile but limited by its tree model

The *Extensible Markup Language* is a standard text format that was published by the W3C in 1998 and updated several times.

His strengths are the following:

- W3C standard, which pushed software vendors to align on the support of the specifications
- Core of a number of related standards to query, validate and transform XML content (XSLT, XML Schemas, XQuery, XProc)
- Endless extendibility, via DTD, XML Schemas or RelaxNG
- Since it's plain text, it's easy to handle with any text editor

XML is based on a tree model. Here is the same family data, but in XML:

```
<families>
 <family familyName="Landais">
  <parents>
   <father id="3">
    <firstName>Eric</firstName>
    <age>34</age>
    <goesToEvents>
     <event href="events.xml#5"/>
    </goesToEvents>
   </father>
   <mother id="4">
    <firstName>Olga</firstName>
    <age>35</age>
    <goesToEvents>
     <event href="events.xml#5"/>
     <event href="events.xml#8"/>
    </goesToEvents>
```

```
   </mother>
  </parents>
  <children>
   <daughter id="2">
    <firstName>Salomé</firstName>
    <age>9</age>
    <goesToEvents>
     <event href="events.xml#11"/>
    </goesToEvents>
   </daughter>
   <son id="1">
    <firstName>Isaac</firstName>
    <age>5</age>
   </son>
  </children>
 </family>
 <family familyName="Todorov">
  <parents>
  ...
  </parents>
  <children>
   <daughter id="2">
    <firstName>Olga</firstName>
    <age>35</age>
    <goesToEvents>
     <event href="events.xml#5"/>
     <event href="events.xml#8"/>
    </goesToEvents>
   </daughter>
   <son id="5">
    <firstName>Boris</firstName>
    <age>42</age>
    <goesToEvents>
     <event href="events.xml#8"/>
    </goesToEvents>
   </son>
  </children>
 </family>
</families>
```

This is one of the many possibilities to structure this information in XML. After thinking a lot, I went for a structure that focuses on parents and children. The problem here is that each person that has children (in this case Olga) needs to be represented twice: as a parent and as a child.

Also, although the events are identified with a unique events ID, it is necessary to know how to resolve the references to get information about the events. it is not self-explanatory.

Storing a flat list of elements, one per person, using IDs to represent all relationships, would address the issue with duplicate entries...

```
<persons>
 <person id="4" parents="10,11" siblings="5" spouse="3" children="1,2">
  <firstName>Olga</firstName>
  <lastName>Landais</lastName>
  <age>35</age>
 </person>
 ...
</persons>
```

...but would result in two annoying issues, the consequences of using XML for a purpose it wasn't designed for:

- slow retrieval of information: since the relationships don't rely on the tree features of XML but on identifiers, for each "hop" between two persons, the whole file would have to be scanned to locate the identifier of the target person.
- That would be like reproducing *the table model* in XML, with its drawbacks.

So, as we know, XML is a very versatile format, but, as we just saw, its tree model doesn't faciliate the cross referencing of information as the only direct relations that XML can express between two nodes of information are the following:

- parent
- child
- attribute

The rest are the result of interpretations which require communication, agreement and extra coding effort.

For instance, using our family information, how do we find the names of Boris' nephews? It is possible, but the query to write is much more complex than the piece of information we try to retrieve. We could also change the structure of the document to adapt it to answer this specific question, but that wouldn't be without a new load of trade-offs to answer other questions.

The RDF graph

The RDF graph model simplifies the interlinking of information.

First things first, the *Resource Description Framework (RDF)*, is a W3C recommendation since 2003, with an update to version 1.1 in 2014. Its purpose is to achieve what the Web has enabled for documents, but this time with data. Since the dawn of the Web, one of its main strength is to enable users to interlink documents easily, building browsing paths for the Web users. One Web page can be the starting point to an infinity of other Web pages. With RDF, it's the same, but with data.

What is a graph?

A graph is a structure in which each node can be related to an infinity of other nodes. Without further ado, let's see what our family data looks like as a graph (not RDF yet):

The graph of the Landais-Todorov family

This graph doesn't represent all the data, but it shows the key features of a graph model: each node can be linked directly to an infinity of other nodes via unidirectional relations. For instance, if we were interested in uncle/nephew relationships and would like to have a more direct connection, we could perfectly link Boris to Isaac and Salomé with an `hasUncle` property, without affecting the rest of the graph. The data model is not limiting.

Another way to represent a graph is write the triples that compose it. A triple is a statement compose of a subject, a property (or predicate) and an object.

Subject	Predicate	Object
Olga	isInLoveWith	Eric
Boris	hasSister	Olga
Salomé	hasBrother	Isaac
...	...	...

From a graph to an RDF graph

RDF is a standard that is used to represent data as a graph, but it also includes a vocabulary to semantically describe the things depicted by the graph. First, let's add a pinch of RDF and semantics on our family graph:

The RDF graph of the Landais-Todorov family

Subject	Predicate	Object
http://elandais.fr/eric	http://xmlns.com/foaf/0.1/givenName	"Eric"
http://elandais.fr/eric	http://example.com/isInLoveWith	http://elandais.fr/olga
http://elandais.fr/salomé	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://example.com/Female
...	...	...

Same triples but in the standard *Turtle syntax*, the most common and convenient way to write RDF by hand:

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix foaf: <http://xmlns.com/foaf/> .
@prefix ex: <http://example.com/family/> .

<http://elandais.fr/eric> foaf:givenName "Eric" ;
 ex:isInLoveWith <http://elandais.fr/olga> .
<http://elandais.fr/salomé> rdf:type ex:Female .
...
```

I have added a lot of new stuff, so I think the best way to explain with clarity is to explain the changes one by one.

What are the xx: prefixes?

If you are familiar with XML namespaces, this is roughly the same in RDF. The prefixes are a shortcut, as everything in RDF is identified with a URL. This way when you type information you don't need to always write the full URI, you only need to declare the prefixes at the beginning of your file. For instance, foaf:age is actually *http://xmlns.com/foaf/0.1/age*, which is a link to the specifications of the Friend Of A Friend vocabulary. This system is very convenient as it enables anyone to create an age property as long as they have control over an Internet domain.

For instance, I could create the property http://colin.maudry.com/properties#age. I haven't created a property for the age of a person, but I have created the DITA ontology, that resides in *http://colin.maudry.com/ontologies/dita#*, and for which the recommended prefix is dita:.

I also could have used ex:age, but FOAF is a very popular vocabulary on the Semantic Web to describe people, and is consequently very well understood by people and software that deal with RDF.

I see URLs instead of people's names

URIs (URL used as identifiers) are used to identify everything in RDF. They are very convenient because they rely on the Web technologies that ensure worldwide validity. If you own the domain yourname.net, you have the control over all the URIs that are based in this domain.

In our data, we see that the Landais family uses identifiers from the domain elandais.fr, because Eric Landais owns the domain. It is also possible to rely on an organization to create identifiers. For instance, I could use the URL of my About.me page to identify

	myself (*http://about.me/ColinMaudry*), since I have some control on what happens when you visit it.
The rectangles are strings, right?	They can be strings, but they can also be integers, dates, or a various range of types. They are groupped under the generic term "literal" and are the only nodes of an RDF graph that don't have a URI. Their specificity is that can only be the subject of a property, not the subject. They are consequently the extremities of a graph.
The graph looks quite messy...	That's what happens when you can link each nodes to any number of other nodes. This way of drawing graphs is only used for small graphs that are useful for future reference. For example, the *DITA ontology* has a graph representation drawn the same way. For small quantities of data you can view it as Turtle. Otherwise, you need to query it.

RDF ontologies

The semantics of the RDF content come from the ontologies.

In order to express semantics in RDF, the practitioners rely on ontologies (also called vocabularies or schemas). Although everyone can write new ontologies about anything and publish them, the most popular ontologies are the ones that solve problems and that are the result of a consensus. This way, software anticipate the expected behavior when processing data based on popular ontologies. One of the most popular ontology in the documentation field is *Dublin core*. It was actually used as a reference in the DITA vocabulary written by the DITA Technical committee at the OASIS.

In RDF, an ontology describes the semantics of certain things through their relations with other things. For instance, if we made an ontology about dogs, we could either focus on the genetic bounds between races, their abilities for certain tasks, the characteristics of their hair, or even their appearance in famous movies. In the end, there is no wrong ontologies, only ontologies that serve different purposes.

DITA for the beginner

DITA is very convenient to manage content as chunks instead of full documents

TODO

The DITA RDF ontology

The translation of the DITA vocabulary into RDF

You can view a graphical representation of the DITA RDF ontology *here*.

Base ontologies

The ontologies from which the DITA RDF ontology is derived

I have chosen to use the Nepomuk ontologies *NIE* and *NFO*, and specialize the classes. The distinction they make between data objects (sequences of bytes) and information elements (interpretations) prevents the confusion between:

- the file as data on a file system, that has a path and can be moved or renamed
- the information contained in this file that results from its interpretation (by an XML editor or a PDF viewer)

For instance, let's consider a DITA topic. If the topic is renamed, can we consider that the topic was modified? The file was modified, but the meaning born by the topic remains untouched.

Publishing method and availability

The DITA RDF ontology is published following the best practices

The namespace base URI is *http://purl.org/dita/ns#* and the recommended prefix is **dita:**.

By default this URI returns the XHTML version of the ontology, but it also supports *content negotiation* for RDF/XML, JSON-LD and Turtle (*Vapour test*).

The ontology is written in a Turtle file and uploaded to an RDF triple store hosted by *Dydra*:

- Turtle (*download*)
- RDF/XML (*download*)
- JSON-LD (*download*)

The Turtle version is the truth. Feel free to checkout the Git project and contribute. A graphical representation of the DITA RDF model can be viewed *on LucidChart*.

The dita2rdf DITA OT plugin

A plugin to the DITA OT that enables export of DITA metadata to RDF

What does it do?

A summary of what the dita2rdf DITA OT plugin actually does

The dita2rdf DITA OT plugin (a.k.a. the dita2rdf plugin) adds a new transtype to the DITA OT: rdf.

In short, running the rdf transtype on a DITA map or topic runs an extraction of the metadata of the content and stores it in RDF/XML format, a serialization of RDF. The transformation not only extracts the metadata of the input file, it also follows the references (@href and resolved @keyref) to cover the whole documentation set.

"Why on Earth would I want to do that?". Good question. I will give you great reasons when I add an extra step to the transformation: the upload of the result to a triple store, an RDF database. I will also speak this project it during the *DITA Europe conference* and the *DITA OT day*. Let's meet there!

Metadata cascading

The dita2rdf DITA OT plugin only extracts the metadata at the level where it's expressed and not to apply the metadata cascading rules in the resulting RDF. I'd rather extract incomplete metadata than zealous inexact statements. The only metadata that is currently inherited is the language, as per the XML standard. For two reasons:

- It is uncommon to mix content from different language in the same documentation set.
- In the cases where languages are mixed, people should specify the language of each topic.

I have ideas to distinguish the metadata that genuinely describes a topic or a map from the metadata that is inherited, but I will not apply them until I'm sure it is robust enough.

For my fellow DITA OT hackers

The dita2rdf plugin was developped reusing the methodology of the pdf2 plugin for the build and the customization, as I thought many DITA-OT users would feel more comfortable with a familiar plugin structure.

However, the dita2rdf plugin differs on a couple of points:

- The preprocess is 'specialized' to remove the steps that are not necessary for the purpose of this plugin. It also speeded up the processing. More details *here*.
- Though the original intent was to rely on the topicmerge to flatten the structure and have only file to process, it appeared too much information was left behind: mapref and @processing-role="resource-only". Topicmerge was consequently also removed, and the parsing is done by following the @href to jump from file to file (the temp files, not the original ones).

For the long term, I aim at removing the preprocess completely and using as much XSLT as possible to make the code more portable.

Requirements

The minimum requirement to install and run the dita2rdf DITA OT plugin

- *DITA Open Toolkit* 1.8.x (version 1.5+ could work but not tested). If you're not sure wich edition to download, get the "full_easy_install", as it contains all the dependencies.

Installing

Requirements, and how to install the dita2rdf DITA OT plugin in the DITA OT

- You have downloaded and extracted the *DITA Open Toolkit*

You want to install the dita2rdf DITA OT plugin in your DITA Open Toolkit.

1. *Download* the dita2rdf DITA OT plugin ZIP archive.
2. Extract `/dita2rdf/dita2rdf-ditaot-plugin.zip` to `[DITA-OT]/plugins`, [DITA-OT] being the directory of your DITA open toolkit installation.
3. Run `[DITA-OT]/startcmd.bat` (or `[DITA-OT]/startcmd.sh` if you run a Linux operating system)A new console window opens.
4. Run the command `ant -f integrator.xml strict` at the root of your DITA open toolkit installation.

The dita2rdf DITA OT plugin is installed in your DITA OT installation and the rdf transtype is available.

Configuring

In order to enjoy the full benefits of the dita2rdf plugin, a bit of configuration is required

The dita2rdf DITA OT plugin must be *installed* in your DITA OT.

You want to configure the dita2rdf DITA OT plugin.

1. In `[DITA-OT]/plugins/com.github.colinmaudry.dita2rdf/customization`, make a copy of `catalog.xml.orig`.
2. Rename the copy `catalog.xml`
3. In `catalog.xml`, uncomment `<uri name="cfg:rdf/config.xml" uri="rdf/config.xml"/>`.
4. Save.
5. In `[DITA-OT]/plugins/com.github.colinmaudry.dita2rdf/customization/rdf`, make a copy of `config.xml.orig`.
6. Rename the copy `config.xml`.
7. In `config.xml`, follow the instructions to edit the configuration.

The dita2rdf DITA OT plugin is correctly configured to extract metadata from your DITA content.

Running the metadata extraction

How to run the extraction of the metadata of your DITA content

You should have *configured* the dita2rdf DITA OT plugin.

You want to extract metadata from your DITA content as RDF.

In [DITA-OT], run `ant -Dargs.input=[path/to/your/ditamap] -Doutput.dir=[path/to/store/the/output] -Dtranstype=rdf`

For instance, on a Windows system: `ant -Dargs.input=C:\Users\colin\dita\map.ditamap -Doutput.dir=C:\Users\colin\dita\output -Dtranstype=rdf`.

The extraction starts.

When you see `BUILD SUCCESSFUL`, the extraction was successful and you can use the RDF/XML file available in the directory you provided for -Doutput.dir.

If you see `BUILD FAILED`, it means there was an error during the extraction. See *Contact and reporting issues*.

Improving the RDF output

Some advice to get more useful RDF output, and possibly fix issues

To avoid issues, here are a couple of things you can do to have good metadata output:

- make sure all the maps and topics have an @id attribute in the root element, and that it is unique in your organization (translations excepted).
- make sure all the maps and topics have a @xml:lang attribute in the root element that respects the recommendations of the W3C. Due to certain spelling and cultural variants (e.g. date formats), distinguishing Bristish English (en-UK) and American English (en-US) is recommended.
- make sure your topicref have a @format attribute with the right value. It can go well without, looking for '.dita' in the @href value, but some URL might abuse this rule (eg. @href="http://blog.dita.xml.org").

Release notes and roadmap

The changes, improvements and bug fixes per version of the dita2rdf plugin + the road map

TODO

Exploring the extracted metadata (a.k.a. "the cool part")

Guidelines to help you explore the extracted metadata

Once the metadata of your content has been extracted and added to a triple store, you can start querying it to get better insights about your content. In the end, the objective is to retrieve enough information to help you roll out your content strategy.

Useful SPARQL queries

A list of SPARQL queries that will help you understand how querying RDF works. Then, you can make your owns!

The *SPARQL Protocol And RDF Query Language (SPARQL)*, is the language used to query RDF data.

Going further, to the infinity and beyond!

DITA metadata is cool, but what about product data? Client data? Support data?

You have tried all the useful SPARQL queries and you have discovered a lot of interesting facts about your DITA content. But this is not the end, it's the beginning.

By now, you know what a graph looks like and have some notions about querying it.

Imagine how it would be to connect other graphs to your DITA metadata graph, such as a product metadata graph, or a company division graph? What about a client graph? The same way you have exported your DITA metadata graph, you could export and connect more metadata from the silos and databases of your organization. You would then be able to query all that data as a single graph and get answers to questions such as:

- Which released products don't have a released user manual?
- For the category of products X, which translations are outdated?
- Which documentation team has the highest rate of topic reuse?
- Could I assemble part or a whole DITA map based on product specifications? (Yes, by writing maps for product features or components, not for whole products! See the *DITA product cascading project*)
- etc...

You could automate the creation of data feeds and reports that would give you the pulse of your organization and help you to take decisions based on tangible facts.

If you are interested in connecting your DITA metadata graph to more data, you can email me at *colin.maudry@semaku.com*, and, together, we will study your requirements.

Contact and reporting issues

How to be in touch and reporting the problems and questions you might have with one of the deliverables

Issues, bugs and questions

Github has a very good issue management system, let's use it!

If you have a Github account, please report the questions and bugs *here* and add the relevant label (**bug** if you get unexpected behaviour of the deliverables, **question** for the rest).

If you don't have a Github account please *send me an email with [dita-rdf] in the subject*.

If you report an issue with the dita2rdf DITA OT plugin, please provide the following:

- the processing logs of the DITA OT
- Your config.xml file
- Your RDF/XML output, if any

Contact

I fluently speak French, English and Spanish.

- Twitter: *@CMaudry* (you can also follow the hash tag *#ditardf*)
- Email: *colin@maudry.com*, or *colin.maudry@semaku.com* for tailored service
- LinkedIn: *Colin Maudry*
- Skype: colin.maudry

Unlicense

The RDF project and its deliverables are licensed under the terms of the Unlicense

The DITA RDF ontology, the dita2rdf DITA OT plugin and all the code included in the DITA RDF project are free and released into the public domain, excepted the parts that were reused from the pdf2 DITA-OT plugin, developed by Idiom Inc.

See *UNLICENSE* and *the Unlicense website*.

If you find the DITA RDF project useful, I would be grateful if you tweet about it (something like "Kudos to @CMaudry for the #DITA RDF project! http://purl.org/dita/rdf-project #ditardf") or if you mention the origin (http://purl.org/dita/rdf-project) and the author (Colin Maudry) of this project in the documentation of your product.