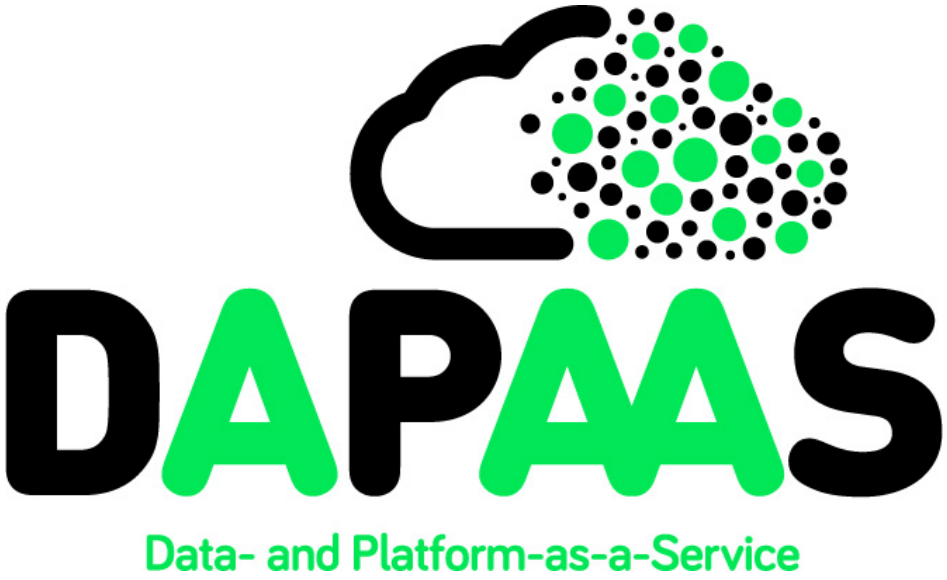
Small or medium-scale focused research project (STREP)

ICT SME-DCA Call 2013
FP7-ICT-2013-SME-DCA

Data Publishing through the Cloud: A Data- and Platform-as-a-Service Approach to Efficient Open Data Publication and Consumption

DaPaaS



Deliverable D4.1

Documented methodology and guidelines

Date:	31.10.2014
Author(s):	Bill Roberts (Swirrl) and Rick Moynihan (Swirrl)
Dissemination level:	PU
WP:	WP4
Version:	1.0

Document metadata

Quality assurors and contributors

Quality assuror(s)	Brian Elvesaeter (SINTEF) and Ivan Berlocher (Saltlux)
Contributor(s)	Bill Roberts (Swirrl), Rick Moynihan (Swirrl), DaPaas consortium members.

Version history

Version	Date	Description
0.1	30.09.2014	Initial outline and Table of Contents (TOC).
0.2	28.10.2014	Complete draft for review.
0.3	30.10.2014	Revision addressing the internal review.
1.0	31.10.2014	Final formatting and layout.

Executive Summary

The main goal of the DaPaaS project is to provide an integrated Data-as-a-Service (DaaS) and Platform-as-a-Service (PaaS) environment, together with associated services, for open data, where 3rd parties can publish and host both datasets and data-driven applications that are accessed by end user data consumers in a cross-platform manner.

An important aspect of DaPaaS is to assist data publishers in converting their data into high quality Linked Data, to maximise the possibilities for effective use of the data.

This document describes:

- A methodology for data publishing, examining in particular the process of transforming data from a range of source formats to Linked Data
- Guidelines on how to apply the methodology
- Requirements for tools to support the methodology
- Design decisions for ‘Grafter’, a software framework for creating Linked Data.

Table of Contents

List of Acronyms

API	Application Programming Interface
CSV	Comma Separated Values (format)
DaaS	Data-as-a-Service
GUI	Graphical User Interface
HTTPS	Hypertext Transfer Protocol Secure
JSON	JavaScript Object Notation (format)
PaaS	Platform-as-a-Service
REST	Representational state transfer
RDF	Resource Description Framework
SLA	Service Level Agreement
SOA	Service Oriented Architecture
SPARQL	SPARQL Protocol and RDF Query Language
SSH	Secure Shell
UML	Unified Modeling Language
XML	eXtensible Markup Language

1 Introduction

This document describes work on a methodology for data publishing in the context of the DaPaaS platform. DaPaaS focuses on use of Linked Data as the main data storage approach and so this methodology concentrates on the process of creating high quality Linked Data.

Linked Data is a highly flexible approach to data representation, which offers benefits to users in terms of combining data from multiple sources. It achieves this through the use of URIs as globally scoped identifiers, use of standard formats and protocols for delivering data, and by incorporating a description of the data structure and meaning in the data itself.

This last aspect makes the use of Linked Data very powerful. However it also places some extra obligations on the user to (a) understand the structure and meaning of their own data and (b) to express this using the relevant standards.

In many alternative approaches to data publishing, for example making files available for download, the structure and meaning of the data are implicitly or loosely defined, so that the user of the data has to read documentation and/or use their own judgement or experience to know how the data can be correctly used. This is an obstacle to use of the data, particularly by other software.

The additional effort for a publisher in producing Linked Data (versus say an Excel file) to represent the data clearly following the standards, only has to be done once, but brings benefits every time the data is used.

However, the DaPaaS project recognises that representing data as Linked Data is an obstacle for data owners. Detailed knowledge of Linked Data is not widespread amongst people with data they want to publish. Therefore data publishers need to be supported by a clear methodology and accompanying tools.

This document sets out the important aspect of that methodology, alongside the main requirements and high level design decisions for the accompanying tools developed within the DaPaaS project. The DaPaaS framework for transforming data into Linked Data is called ‘Grafter’.

This document addresses the DaPaaS requirement on data publishing methodology – requirement “DP-13 Data publishing methodology support” as outlined in Deliverable D2.1. This overall requirement has been primarily addressed by analysing a set of data publishing requirements and scenarios, and the development of Grafter.

The remainder of this document is organised as follows:

- Analysis of the requirements for a data publishing methodology
- Scenarios that the methodology should support
- Review of existing approaches and tools
- Description of the Grafter framework to support the methodology
- Summary and outlook

This document is accompanied by more detailed online information, provided as part of the Grafter documentation and covering both guidelines for transformation of data to Linked Data and detailed instructions and examples on how to achieve that using the Grafter framework. The online documents are available at:

- http://grafter.org/guidelines/index.html
- http://grafter.org/howto/index.html

2 Requirements Analysis

2.1 High level requirements

The main objective of Linked Data publishing is to make the chosen data as useful as possible for the users. To do this well transformation and cleaning is usually necessary and there is scope for users to introduce additional errors into the data. Therefore it's important for the methodology to provide guidance on best practice patterns for data modelling, along with processes and approaches that emphasise validity, reliability and correctness.

The data produced by following the methodology should:

- be free from errors
- be consistent: both self-consistent and consistent with common practices of the domain of interest
- be well-documented and self-documenting: the user should be able to clearly understand the data, using only the data and accompanying metadata
- be standards compliant: the data should follow widely accepted standards wherever possible
- use common points of reference: by using commonly used externally maintained identifiers for reference data, where it exists
- link to other data: make it clear how this data relates to other collections of data that potential users are likely to want to use.

The methodology must be accompanied by tools to help data publishers achieve these goals.

The objective for the DaPaaS platform is that it should support a wide variety of data types, therefore the methodology and supporting tools must also be generically applicable.

2.2 Steps in the data publishing methodology

The starting point for our analysis is the assumption that the data has already been collected and is in some sort of order. It is likely to be in data formats and software systems designed either for data collection or for internal use of the data.

The process of Linked Data publishing generally involves the following steps:

1. Review the available data.
2. Understand what it means and how it is structured.
3. Determine whether the data is consistent and check for missing data or errors. The data may need to be filtered or cleaned.
4. Identify the main entities of interest within the data.
5. Choose a system of URIs for referring to those entities. In some cases this may involve choosing from existing URI sets. In others it will involve designing new URIs. In principle these can be any valid URI, but in practice there are a number of common patterns which can be followed. For detailed guidance, see
   - http://patterns.dataincubator.org/book/identifier-patterns.html
   - http://www.w3.org/2011/gld/wiki/Linked_Data_Cookbook#Step_2_Name_Things_with_URIs
   - http://linkeddatabook.com/editions/1.0/#htoc36
6. Consider how the entities are related to each other and which of their attributes it is important to represent. Sometimes this is obvious – in other cases more complicated patterns might be required. See http://patterns.dataincubator.org/book/modelling-patterns.html for explanations of commonly occurring patterns in Linked Data representation.

7. Express the structure of the data (the 'data model') in terms of RDFS or OWL classes and properties. Look for existing vocabularies that are a good match for the meaning of the data. If a well-documented maintained vocabulary exists that meets your needs, then by using it your data can be easily connected to similar data from other sources. Vocabularies that have been through the W3C or other standardisation process are ideal, but vocabularies generated by individuals or groups for their own needs can often be useful too. Discovery of suitable vocabularies can be challenging. Looking at other data on existing topics is often a good way to do it. Services such as 'Linked Open Vocabularies' (http://lov.okfn.org/dataset/lov/) are also useful. In many cases, no pre-existing suitable vocabulary will exist and the data publisher must create their own. Ensure that any such vocabulary is well-documented following standard approaches. The definitions of the vocabulary terms should be included in the data to be published, so that others can find it.
8. Having decided how the data should be represented, transform it from its existing format to its target Linked Data representation. (This is the core function of the Grafter framework). Almost all datasets are too large to do this by hand, so a software based approach is required.
9. Check that the output of the transformation is correct. This may involve a combination of manual inspection and automatic testing. A useful approach is to define SPARQL queries that check conformance to particular patterns. These can be designed to return information about failures to match the required data structure (see http://danbri.org/words/2005/07/30/114 , https://github.com/ldodds/sparql-check and http://www.w3.org/TR/vocab-data-cube/#wf for discussion and examples).
10. Add metadata to help users know how they can use the data. This typically includes a licence, information on when the data was updated, who published it, what it is about. There are standard vocabularies for Linked Data metadata, the most common of which are Dublin Core (http://dublincore.org/documents/dcmi-terms/), DCAT (http://www.w3.org/TR/vocab-dcat/) and VoID (http://www.w3.org/TR/void/).
11. Load the data into the publishing system of your choice.

This process[1] is relatively complex and involves multiple steps which leads to it being both error prone and requiring specialist knowledge. Consequently it is common for additional errors to be introduced and propagated downstream after publication.

The purpose of Grafter is to reduce the amount of specialist knowledge required whilst reducing the likelihood of propagating errors by automating manual steps, and introducing mechanisms for data validation and cleaning.

[1] See also Chapter 3 of the Euclid Project learning materials for another perspective and more detail on the same fundamental steps: http://www.euclid-project.eu/modules/chapter3.

3 Methodology Scenarios

Some representative examples of common data types that users typically want to publish as Linked Data are as follows:

1. CSV file or spreadsheet with consistent rows (optional header row plus lots of uniformly structured data rows)

   This is the case of a 'simple' tabular file. (Note that the W3C 'CSV on the Web' working group is addressing this case in some detail and has produced a series of use cases – see http://www.w3.org/2013/csvw/wiki/Main_Page)

2. One record per line, with record type indicator

   Another common pattern, especially in CSV files, is a line-based structure, where the value of the first column is a 'record type' indicator, that tells you how you should interpret the rest of that line.

3. Spreadsheet with non-uniform structure

   Many data containing spreadsheets do not follow the simple table structure of scenario 1. A spreadsheet can incorporate multiple worksheets. Each worksheet sometimes includes several tables. It is common to have more than one header row, as multiple dimensions of data are flattened into a series of columns. There are often empty rows, or rows containing summary or 'total' data. And there is often commentary or metadata incorporated in the file.

4. Relational database - one or more queries to extract data, then convert to RDF

   Relational databases are a common data source for linked data. The R2RML language (http://www.w3.org/TR/r2rml/) has been designed to specify the transformation of relational database data to RDF. Other conversion methods are based on using SQL queries to obtain a data extract. In other cases, parts of relational databases are exported to CSV. In some cases, individual tables can be covered by scenario 1, but if the exported tables are related (via relational foreign keys), then often they need to be considered together when creating an RDF representation.

5. ESRI Shapefile or other geographic format

   There are several common formats of geographical data, all structurally reasonably similar: ESRI Shapefile, GML, KML, GeoJSON - consisting of a set of geographical features, with attributes and a geometrical description, typically consisting of points, lines, or polygons.

6. Web form or interactive application.

   The user fills in form fields, or takes some other action in the application such as selecting from a list or clicking on a map, which leads to creation of RDF behind the scenes.

7. Copy of external RDF

   Sometimes we want to copy selected RDF from an external source into a data store. The simplest case is downloading and uploading data. It can also be done by running a CONSTRUCT query to obtain selected data from an external source. A further step can be to use the CONSTRUCT query to apply transformations to the source data.

8. Extraction of data from an API

   Many data sources provide an API. (Access to data via a SPARQL endpoint is covered by scenario 7. Here we are thinking of APIs providing data in non-RDF formats). Code can be run to obtain selected data from an external API, then transform it to linked data and load it to a data store.

All of the above scenarios involve creating an initial data transformation process, following the steps defined in Section 2.2.

In many cases, it is required to keep published Linked Data up to date, by processing updated versions of the source data, then replacing or appending to the Linked Data dataset. This has its own issues that should be addressed by supporting tools, such as: validating that the structure of the source hasn't changed, so that the results of transformation code are reliable; knowing whether the update should involve deleting some existing data as well as or instead of adding new data.

4 Review of Relevant Data Transformation Tools

The process described above falls into the general class of 'Extract, Transform, Load' (ETL) processes. This is a common and long-standing problem in information technology and there are many existing software tools.

Before deciding to develop the Grafter framework, a number of existing software tools were reviewed. An existing tool that could be used or adapted would be preferable to developing new software, providing one could be found that met our requirements.

A candidate tool would need to support the data publishing methodology described above. The ETL process is most relevant to step 8 of the methodology, the data transformation part, but that step must operate within the context of the whole process and be well suited to producing Linked Data as its output format.

It would also need to work for the range of scenarios set out in Section 3.

Those needs correspond to more specific requirements:

- well-suited to producing RDF as the target output
- possible to integrate into the DaPaas platform
- already have a Graphical User Interface (GUI), or be suitable for one to be added
- ability to use via an API, so that it can be automated and incorporated into other software tools
- ability to serialise, export, version control and exchange transformation definitions
- ability to accept a range of input types
- perform well with large datasets, both via API and via the GUI

A number of existing ETL tools were reviewed, including OpenRefine, Pentaho (Kettle), Talend Open Studio, Taverna, Orange Data Miner, Weka 3.

Of these, OpenRefine, with the RDF plug-in developed by the DERI institute of the National University of Ireland Galway, came closest to our requirements. It has a good interactive user interface, a familiar tabular UI design and it is excellent for manual clean-up and conversion of small to medium size datasets.

After detailed review of the capabilities and code of OpenRefine, we concluded that its core design was not well suited to robust ETL processes. The core code is tightly coupled with the UI and workspace concept, which makes it very difficult to extract the transformation parts for automation. The code base is not componentised which limits the ability for code re-use. Transformations can be shared but the process is brittle and the inefficient multi-pass approach to executing transformations limits the performance of the tool for large datasets.

Our conclusion was that we should learn from and take inspiration from OpenRefine and the good points of other tools we came across, but to build something which would meet our specific needs.

5 DaPaaS Tool-Supported Methodology: Grafter

Grafter is an open source framework for transforming data to RDF. The code is available at https://github.com/Swirrl/grafter and documentation and examples are available at http://grafter.org.

It is still undergoing development but Swirrl has been using it in several commercial projects where it has already converted over 300 diverse datasets, generating almost 200 million production triples. Additionally SINTEF is working on a graphical user interface for it (see deliverable D2.2) and Saltlux has been using it for the DaPaaS use cases.

Grafter aims to incorporate the following:

- Grafter Domain Specific Language (DSL) a basic framework based on functional programming, laziness and immutability, useable by programmers, for machines to read and write
- Grafter Import Service: uses a pluggable Grafter transformation, builds a user interface form, receives data, transforms it and loads it; reports meaningful errors back to the user
- Grafter Pipeline Builder user interface: interactive tool to help users build pipelines; run them in the importer, or mix with custom code; spreadsheet-like UI with real-time feedback

It is designed to meet the requirements set out in Section 4, and to be well suited for use with the Methodology described in Section 2.2.

It is developed in Clojure, a functional programming language and Lisp dialect which runs on the Java Virtual Machine. Clojure treats code as data, which means that transformation pipelines can be easily serialised, put under version control and shared between users. The fact that Clojure uses the Java Virtual Machine means that Clojure programs can be easily integrated with Java libraries, giving access to the wide range of useful existing code bases for working with RDF, office automation software and so on.

Important design choices include:

- Designing a library of simple transformation functions that work on a cells, rows or tables of data and use those functions to build a domain specific language for transforming data to RDF.
- Ensure those primitive functions can be composed into a ‘pipeline’.
- The use of Turtle-like graph templates to convert row level data into RDF by intuitively mapping columns into the graph.
- Do not support complex looping and branching in the DSL or user interface, as this makes transformation pipelines hard for users to build and hard to debug. If more complex logical structures are required, functions from the DSL can be called from another program written in Clojure or Java.
- Ensure that clear error reporting is possible.
- Make use of Clojure’s ‘lazyness’ to support efficient processing of streaming data.
- Avoid use of intermediate formats where possible.
- Ensure that data cleaning can be part of a transformation pipeline as well as the data transformation parts. Data cleaning can be time consuming and it is important that the processes carried out to tidy up source data are controlled and repeatable.

Grafter documentation is available at http://grafter.org and includes:

- Detailed API documents: http://api.grafter.org/0.2/
- How-to guides: http://grafter.org/howto/index.html
- Detailed worked examples: http://grafter.org/example/index.html
- Guidelines on use with the DaPaaS methodology: http://grafter.org/guidelines/index.html

6 Summary and Outlook

A process for transforming and publishing Linked Data has been defined and documented – and used to specify the Grafter tool framework to help users through the process.

The Grafter software has been developed to an effective operational stage, though further development is required and ongoing. It is already an extremely useful tool for those with reasonable linked data and programming knowledge to carry out data transformations more quickly and more reliably.

Steps to add a Graphical User Interface to Grafter are in progress. Provision of a friendly and robust user interface is an essential step in assisting a broader group of users, with less specialist knowledge, to create high quality Linked Data with a reasonable level of effort. Work in Year 2 of DaPaaS is planned to take Grafter to this next stage.

Bibliography

Euclid Project Learning Materials, http://www.euclid-project.eu/resources/learning-materials

Linked Data Book (Heath and Bizer), http://linkeddatabook.com/editions/1.0/

Linked Data Patterns (Dodds and Davis), http://patterns.dataincubator.org/book/

Linked Data Cookbook, W3C Government Linked Data Working Group.
http://www.w3.org/2011/gld/wiki/Linked_Data_Cookbook