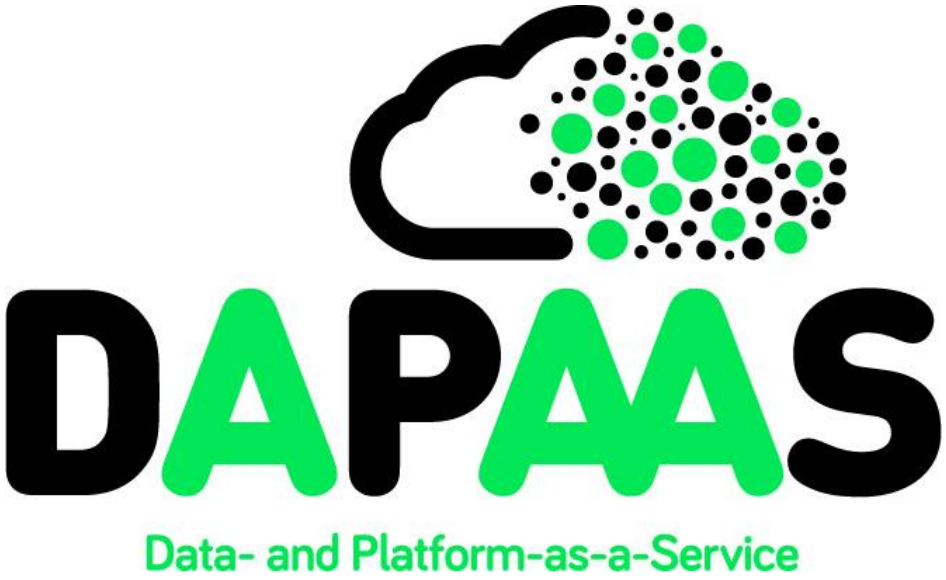
Small or medium-scale focused research project (STREP)

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 D4.2:

Software tools integrated into platform

Date:	30.04.2015
Author(s):	Bill Roberts (Swirrl)
Dissemination level:	PU
WP:	WP4
Version:	1.0

Document metadata

Quality assurors and contributors

Quality assuror(s)	Nikolay Nikolov (SINTEF), Alex Simov (Ontotext)
Contributor(s)	Bill Roberts (Swirrl), Rick Moynihan (Swirrl), DaPaaS consortium members.

Version history

Version	Date	Description
0.1	28.04.2015	Complete draft for review.
0.9	29.04.2015	Final version.
1.0	29.04.2015	Title corrected.

Executive Summary

A key problem in delivering a web based data as a service platform for open data is making it easy for users to host their data, import it, and present it in a way that makes sense for others and the web itself.

Linked Data's biggest benefit is in representing data in a way that is designed for the web. Data points and the vocabularies for modelling them need to be represented by URIs. Using URIs allows data (and vocabularies) to be linked together in the same way that the web lets you link documents. One of Linked Data's primary benefits is in how it lets you globally connect data together.

One of the primary concerns of the DaPaaS project is to take the data that people have already, which is very often tabular, and transform it into data for the web – which can then be hosted on the DaPaaS platform.

The project has already developed a Methodology for transforming data into Linked Data, documented in project deliverable D4.1. This document describes the software that has been developed to support that Methodology. It consists of two main components: Grafter, a system for defining and executing data transformations, and the Graftwerk framework for supporting integration of Grafter transformations into the overall DaPaaS platform.

Table of Contents

List of Acronyms

API	Application Programming Interface
DSL	Domain Specific Language
PaaS	Platform-as-a-Service
REST	Representational state transfer
RDF	Resource Description Framework
SOA	Service Oriented Architecture
SPARQL	SPARQL Protocol and RDF Query Language
UI	User Interface

1 Introduction

The DaPaaS project has developed a Methodology and accompanying guidelines on publishing data as Linked Data: with the objective of making the data accessible and re-usable in both human-readable and machine-readable forms. The Methodology is documented in Deliverable D4.1 "Documented Methodology and Guidelines".

The project has developed a cloud-based, platform-as-a-service software platform for transforming, storing and providing access to data, with working title Datagraft. The overall architecture of Datagraft has been described in Deliverable D2.2 "Open Data PaaS prototype v1".

The work described here fits primarily into the "Data cleaning and transformation" component of the overall architecture. This document describes the software used to support and enable those data cleaning and transformation functions, and the integration of this with the Datagraft platform.

There are two main aspects to this: the Grafter system for defining and executing data transformations, and the Graftwerk framework for supporting integration of Grafter transformations into the overall Datagraft platform. Graftwerk enables 'Transformation as a Service' – essentially providing API access to various Grafter-powered functions.

Other components of Datagraft are responsible for providing a user interface for editing and executing the data transformations powered by Grafter and for managing the relevant input and output data. Those components are described in other project deliverables.

2 Requirements Summary

High level needs such as making data appropriate to the audience consuming it, and to the web medium it lives in, usually necessitate some form of transformation process.

Transforming data is critically important to being able to derive utility from it, but in any transformation process there is a risk that additional errors may be introduced into the data. Consequently in addition to methodologies to help formalise processes around transformation and error mitigation there is a clear role for interactive tools to automate repetitive and error prone tasks, and for them to do so in a way that minimises the likelihood of introducing errors.

The vast majority of data that users are skilled in handling is tabular in nature, and is typically manipulated in spreadsheets, or relational databases. These tools have been common place since the 1980s and users know how to use them.

This poses a familiarity challenge when transforming the data into the graph of connections required for linked data. For this reason we believe a strong requirement is to support this transformation process through a familiar spreadsheet-like interface.

Additionally we have identified two distinct user groups, Knowledge Workers and Software Developers, with divergent requirements who both have important expertise that need to be leveraged if we are to enable the collaboration and seamless workflows that are required to power the open data revolution.

2.1 Supporting Knowledge Workers

The first and perhaps most important group of users to empower are the knowledge workers, who are used to handling and processing tabular data in tools such as Excel. These users are typically domain experts, skilled in manipulating the tabular data they are custodians of. This demographic provide domain expertise, knowledge of statistics and particular datasets. They often know how the data was assembled and what conclusions and techniques can be appropriately applied to the data.

Typically these users spend a large amount of time repeatedly applying the same transformations to data. For example generating updated figures for a monthly report might require tedious, error prone, manual transformations to be applied to the dataset every month.

We believe that these users are best served by an interactive graphical user interface that presents a familiar spreadsheet-like interface, focused on extracting, transforming and loading data.

User interfaces to enable non-developer programming have been a holy grail in computing since the early 1980s. However interfaces which present their users a Turing complete[1] programming model have largely been unsuccessful. This is primarily because they become simultaneously too complicated for non-developers to use and too restrictive for developers.

The solution to this is to simplify the programming model presented through the interface to explicitly try to avoid Turing completeness.

This approach has enjoyed wide success, leading to innovations such as spreadsheets (a restricted form of data-flow programming), musical synthesisers and trackers, email filtering, as well as simple tools for recording and automating image processing tasks.

The most successful end user programming tools prevent constructs for complex control flow, such as looping, recursion and branching statements in order to prevent users from the complexities and problems that go with being able to express all computable transformations.

We believe that there is a fertile middle ground where users can have both an intuitive and familiar user interface and just enough expressivity to be able to transform the majority of the datasets they encounter. Importantly this constraint should help guarantee properties such as termination, which for example make it impossible for users to develop programs that get stuck in infinite loops.

2.2 Supporting Software Developers

The second segment of target users is software developers. Developers are important because they have the expertise to build bespoke complex transformations that cannot be expressed through the UI or by domain experts. Also developers may be required for transformations that require high levels of validation, integration and automation, or need additional tool support.

This user group may lack some of the domain knowledge of the experts, but they have the experience required to produce and debug far more complex transformations - transformations that can be produced only via a Turing-complete programming language.

We believe that this audience are best served by well documented APIs that are compatible with the user interface, and that the best way to achieve this is to ensure that the UI uses the same APIs developers do.

In order to build additional tooling and support for more complex transformations developers demand a clear separation of responsibilities between system components. So the APIs for transforming data should be cleanly separated from other superfluous components such as the user interface. This is a significant problem with the OpenRefine system, where the user interface is tightly coupled to its transformation engine, making it hard for developers to make use of the transformation engine in other software.

2.3 Summary of technical design criteria

In order to allow an efficient solution to the requirements of our main user groups, a series of high level design criteria were developed. A solution to the requirements should incorporate:

- modular, re-usable components to carry out specific parts of the transformation process;
- the ability to manage the entire transformation tool-chain using standard software version control approaches;
- the ability to process large data collections (hundreds of millions of triples) at reasonable speed, without being limited by available system memory;

[1] http://en.wikipedia.org/wiki/Turing_completeness

- the ability to incorporate transformations into other software systems, to enable automation of data transfer between various systems.
- exploitation of existing software libraries where available, for example to parse and process input files of various types, or to generate RDF in different formats.

2.4 Review of existing alternatives

A number of existing approaches to similar data transformation requirements were reviewed, including both RDF-specific tools and generic Extract-Transform-Load (ETL) tools. These included R2RML[2], TARQL[3], OpenRefine[4] (with the DERI RDF extension) and Pentaho/Kettle[5] .

Whilst all of these had their own strengths, none were viewed to be a sufficiently good match to the specified criteria.

The template-based approach of R2RML is attractive, but a declarative approach becomes inflexible when more complex data transformations are required. Our view was that to support the range of necessary transformations we would need to call on the power of an existing full-featured programming language. A detailed assessments of R2RML implementations was not carried out. The suitability for driving a user interface using R2RML would depend on the details of the implementation.

TARQL uses SPARQL CONSTRUCT queries to generate RDF from tabular input. This approach is appealing in the way it makes use of standard SPARQL, but any non-trivial programming becomes cumbersome within a SPARQL query. Furthermore the approach suffers from limitations related to the cost of performing graph matching using SPARQL over large datasets.

The Kettle system from Pentaho is a powerful general-purpose ETL system, with many available plug-ins and components and an efficient row-based processing system. The negative aspects, when measured against the specific criteria that have been defined, were lack of existing RDF support, and a user interaction model that is essentially too powerful. Kettle provides a ‘Turing complete’ approach with conditional operators, loops and recursion. This makes debugging and error reporting hard and makes it very difficult for a server to be sure if a transformation is ‘safe’ and presents a complex user interface to users. While we want a powerful programming language behind the scenes, to construct the aforementioned re-usable modules, the way in which a user combines these modules together to create a pipeline should be kept simple.

OpenRefine was the candidate system that came closest to meeting our needs. The graphical user interface of OpenRefine is easy to understand and use. The perceived limitations for our needs were around the difficulty in separating the core implementation from the user interface, so presenting challenges to meet our requirements for componentisation and automation. Also, it involves a multi-pass design which can become slow for large datasets.

3 System Design & Integration

There are two primary technology outputs developed by Swirrl and deployed as supporting infrastructure on the Datagraft platform. The first is Grafter; an open source domain specific programming language for expressing data transformations. Grafter provides the library of primitive operations for transforming tabular data into linked data, along with a suite of utilities useful for developers working on ETL (Extract, Transform and Load) systems.

The second technology, Graftwerk, exists to integrate user created Grafter transformations into the Datagraft platform at the service layer. Graftwerk implements the "transformation as a service" component, providing a RESTFUL service that can execute Grafter transformations on source data.

[2] http://www.w3.org/TR/r2rml/
[3] http://tarql.github.io/
[4] http://openrefine.org/
[5] http://community.pentaho.com/projects/data-integration/

Graftwerk delivers two key features to the Datagraft platform:

- Executing Grafter transformations, applying them to the source data and returning the resulting transformed data in the desired format (either tabular or RDF).
- Applying Grafter transformations to a preview of the data, allowing a paginated preview of the transformation formatted for display to the end user.

The Grafterizer workflow engine and user interface, developed in Work Packages 2 and 3, uses the underlying Graftwerk service to provide both its real time preview functionality and the capability to execute transformations on whole datasets.

Grafter, Graftwerk and Grafterizer combine to implement the "Data Cleaning & Transformation" and "Data Workflows" components of the overall architecture of the DaPaaS Datagraft platform. Figure 1, copied from Deliverable 2.2 ("Open Data PaaS Prototype v1"), shows how these components fit with the other parts of the parts of the PaaS architecture, as developed in WP2, and indicates that these components make use of the Data Layer APIs as developed in WP1. The Grafter and Graftwerk activities of WP4 are hence integrated with the outputs of WP1, WP2 and WP3.

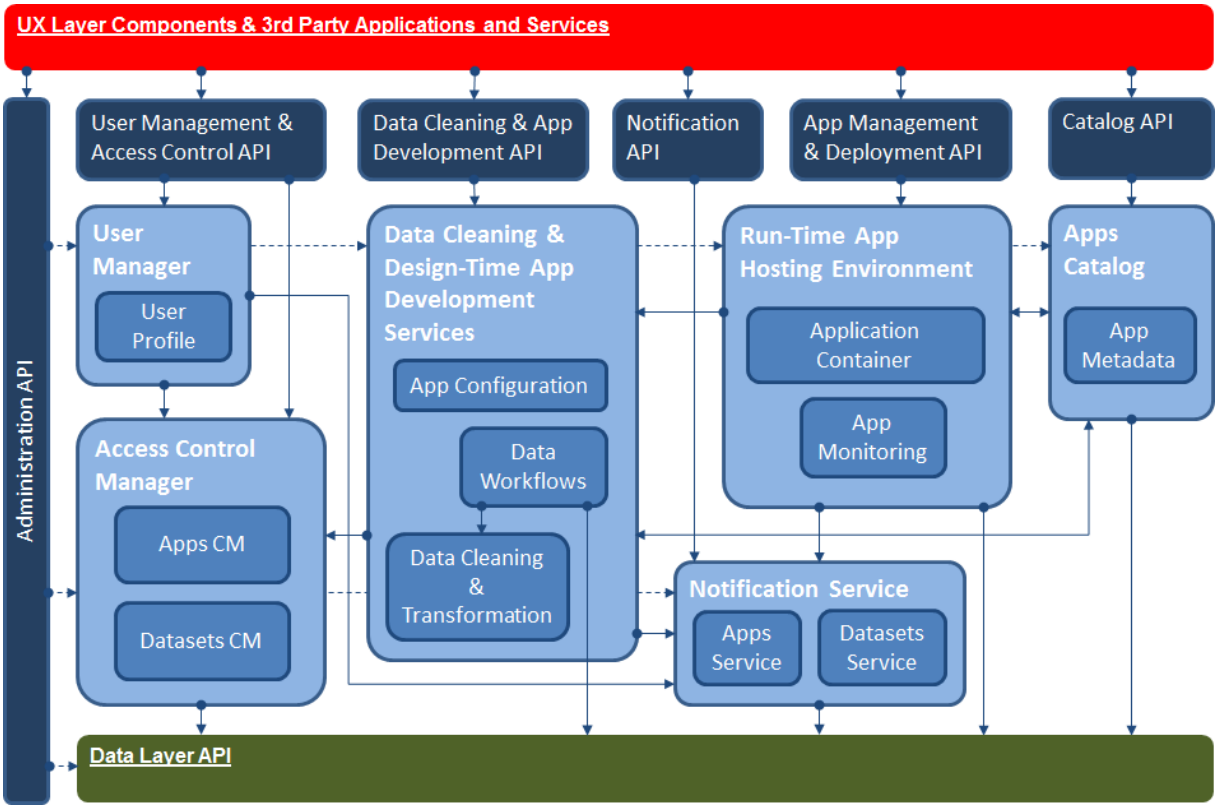


Figure 1 Overall architecture of the PaaS layer, copied from D2.2

The overall architecture of the Grafter components and how they interact with the relevant stakeholders is shown in Figure 2:

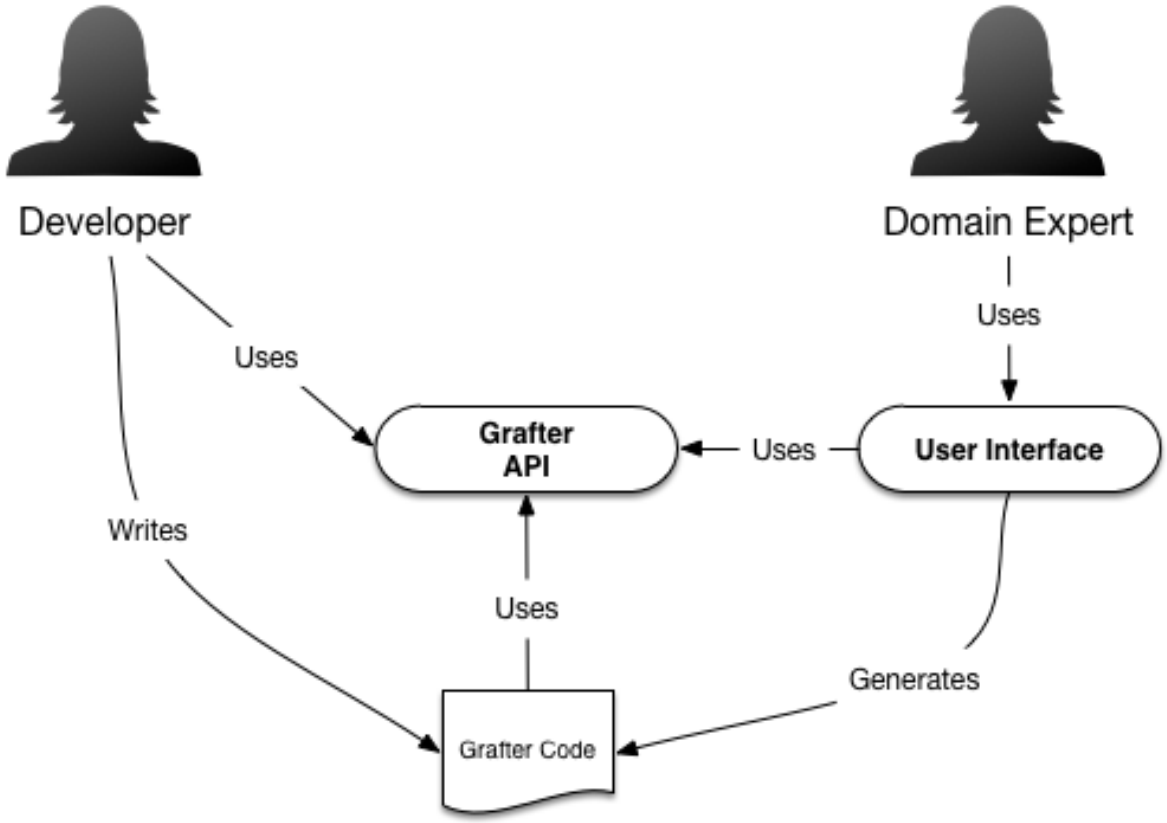


Figure 2 Grafter Components

3.1 Grafter

Grafter[6] is an open source API and suite of software tools created by Swirrl in the DaPaaS project to support complex and reliable data transformations.

It is implemented using Clojure[7], a functional programming language that runs on the Java Virtual Machine (JVM). A functional approach is well suited to the idea of a 'transformation pipeline' and by using a JVM-based system, it becomes straightforward to exploit the large collection of useful libraries already available in the Java programming language.

Throughout its development, Grafter has been used by Swirrl to help perform large scale ETL jobs, to thoroughly test the framework by using it for practical tasks, and to expose new requirements. It has been applied to converting hundreds of datasets of tabular data into Linked Data.

Grafter benefits include:

- It supports a clean separation of concerns from both graphical tools for building transformations and import services.
- Transformations are implemented as pure functions on immutable data, which makes the logic of the transformation process significantly easier to reason about.
- It has been designed to help support interactive user interfaces for developing and testing transformations.

[6] http://grafter.org
[7] http://clojure.org/

- It supports large data transformations efficiently. Unlike some other tools it takes a streaming approach to transformation, which means the maximum size of dataset that can be processed is not limited by available memory.
- It supports an easy way to convert tabular data into Linked Data, via graph templates.
- It has an efficient streaming implementation of a normalising melt operation (going from a 'wide' table format to a 'long' table format), that lets you easily transform cross tabulations featuring arbitrary numbers of categories (frequently used to summarise data), back into a normalised representation suited for machine processing. A common use case is in converting pivot tables.
- It provides APIs for serialising Linked Data in almost all of its standard serialisations.
- It provides integration with triple stores via standard interfaces.
- It has a highly modular design.

3.1.1 Grafter Design

Grafter is broken down into a variety of modules in order to cleanly demarcate functionality, as illustrated in Figure 3. These modules broadly fall into two categories identified by the namespaces `grafter.rdf` and `grafter.tabular`.

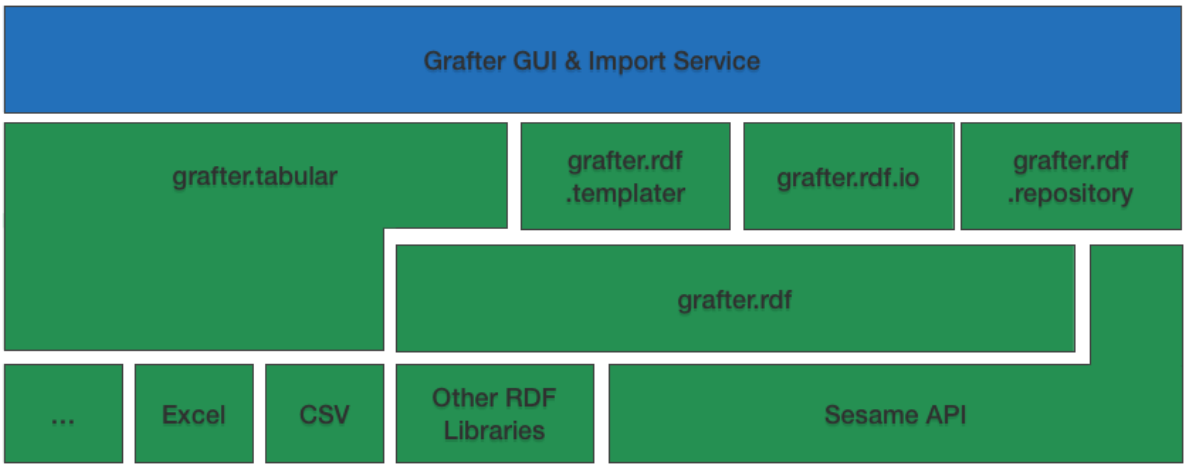


Figure 3 Grafter Architecture Stack

These two primary divisions represent the two sides of the ETL problem Grafter is trying to solve:

- The cleaning and transformation of tabular data.
- The conversion and loading of that data into Linked Data (RDF).

The `grafter.tabular` namespace contains a wide variety of useful functions for filtering data by row, column or cell contents; and applying arbitrary transformation functions to cells through functions like `derive-column`. Additionally more complex and important functions for normalising data into a more uniform form exist such as `fill-when` and `melt`.

Functionality is also being added to help materialise errors and ensure they can be displayed in the appropriate cell or context where they occur.

3.1.2 Pipes

Tabular Grafter transformations are typically expressed as a sequence of stepwise operations on whole tables of data. All tabular operations are simply pure functions from a `Dataset` (a table) to a `Dataset`.

This can be seen in the example Grafter code below:

```
(-> (read-dataset data-file)
    (make-dataset [:area :name :period "measure" "lcl" "ucl"])
    (drop-rows 1)
    (melt :area :name :period)
    (derive-column :area-uri [:area] statistical-geography)
    (derive-column :alc-hosp-uri [:variable :period :area] alc-hosp-id))
```

This tabular dataset transformation processes a spreadsheet containing hospital admissions due to alcohol in Glasgow, a final step (not yet shown) converts it into Linked Data.

Each line is a function call that receives a `Dataset` (table) and returns a new one that has been transformed. Sequences of tabular operations such as this, where a table is received as input and returned as output, are called pipes. When operations are composed together like this on tabular datasets, we refer to them inside Grafter as pipes.

Pipes are just one or more pure functions composed together with the restriction that they receive a `Dataset` as their first argument, and must return a Dataset as their return value.

Figure 4 Pipe composition

The interesting property about pipes is that they can be composed together arbitrarily, and you'll always get another pipe. Additionally because the inputs, outputs and intermediate steps to pipes are always tables; they are very intuitive for users to manipulate and use.

3.1.3 Grafts

In order to publish Linked Data a final transformation must take place into the graph structure[8] used by RDF. To achieve this, a final step is required that maps each row of the source data into a graph. That graph is made up of a sequence of 'quads' as used in the RDF approach, each consisting of a subject, predicate, object and context.

[8] http://en.wikipedia.org/wiki/Graph_%28abstract_data_type%29

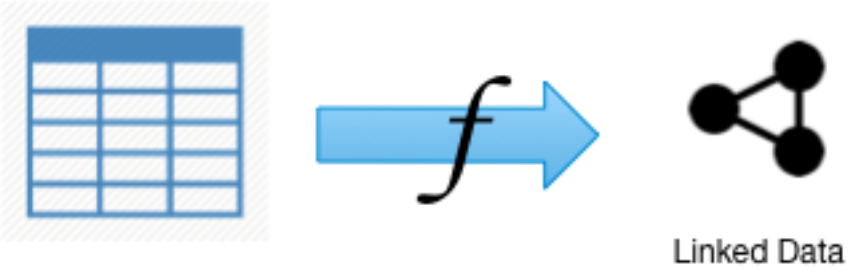


Figure 5 Creating linked data

Because a Graft takes a table and returns a lazy sequence of quads representing the linked data graph, they don't have the composition property that pipes do. However, additional filtering steps can be added to the stream of quads if necessary.

Typically the bulk of transformations are best performed whilst the data is in the table, though post processing can be performed by filters.

Grafter supports a simple graph template to express the conversion of each row of an input tabular dataset into a graph. That template makes use of a simple Domain Specific Language (DSL) to specify commonly used data transformation functions, combined with selections from the input data and literal values.

```
 (graph-fn [{:keys [alc-hosp-uri area-uri name period variable value]}]
            (graph (base-graph "health/alcohol-hospital-admission")
                   [alc-hosp-uri
                    [rdf:a "http://purl.org/linked-data/cube#Observation"]
                    [rdfs:label (rdfstr (str "Alcohol hospital admissions, "
                                             variable ", "
                                             period ", "
                                             name))]
                     [(alc-hosp variable) (parseValue value)]
                     [qb:dataSet  "http://linked.glasgow.gov.uk/data/health/alcohol-
hospital-admission/"]
                     [scot:refPeriod (year-prefix period)]
                     [scot:RefArea area-uri]]))
```

The above code is used to express the mapping of columns in a tabular Dataset into its position in a Linked Data graph.

3.2 Graftwerk

The DaPaaS Datagraft platform integrates an array of components and technologies from consortium members, and 3rd parties such as Amazon. Due to the wide range of technologies and organisations, each with their own standards and deployment processes, a Service Oriented Architecture (SOA) approach to design was taken.

One of the primary means of integration within the Datagraft platform is via RESTful API services for the major components. Consequently other Datagraft sub-systems, such as the transformation builder

(Grafterizer) and the catalog, need to integrate with Grafter via a RESTFUL service. This is the role of the Graftwerk service.

Graftwerk provides a sandboxed execution environment for Grafter transformations and supports two primary platform features:

1. The ability to execute a supplied Grafter transformation on the entirety of a supplied tabular dataset. The results of the whole transformation are returned.

2. The ability to specify a page of data in the tabular data to apply the supplied Grafter transformation to, and to return a preview of the results of the transformation on that subset.

The first of these features ensures that transformations hosted on Datagraft can be applied to arbitrary datasets, generating results for download or hosting. The second feature for generating live previews of the transformation is critical to provide a high quality interactive user experience via the interface. Graftwerk supports both of these features on both kinds of transformation: pipes and grafts.

3.3 Graftwerk Service Design

Building the kinds of cohesive and integrated experiences that users expect, always leads to an integration challenge; especially when integration involves collaboration across consortium and organisational boundaries, and there is a desire to reuse work in other contexts.

These desires for tight integration and reuse are often fundamentally at odds, especially if the services are stateful. Graftwerk itself integrates with the Datagraft and other platforms in a completely stateless manner, thus reducing coupling between components, enabling reuse and simplifying the architecture.

The RESTFUL interface Graftwerk exposes is therefore effectively also a pure function, that receives essentially two arguments, the transformation code to execute and the data to operate on, as shown in Figure 6.

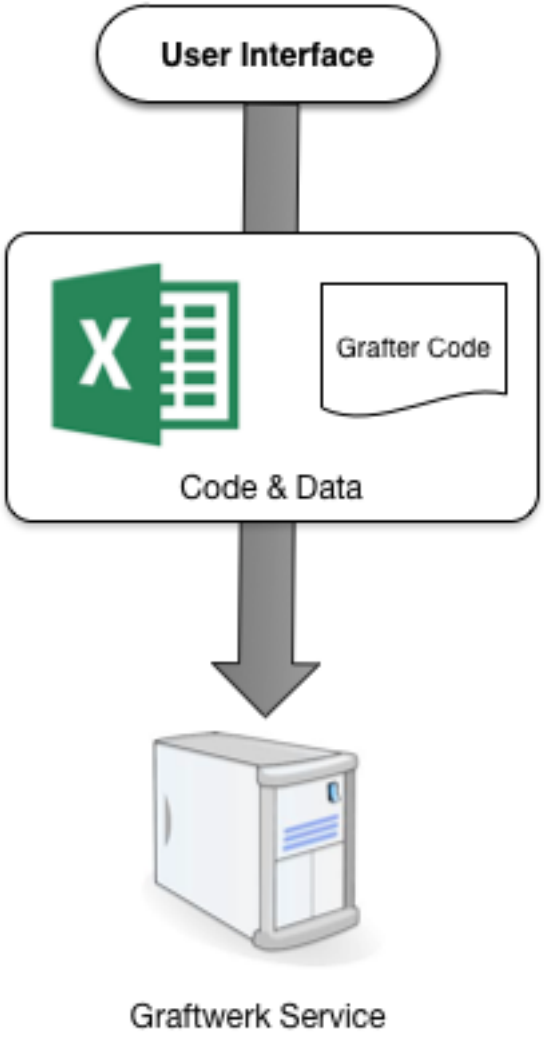


Figure 6 Request to the Graftwerk service

In response Graftwerk applies the supplied transformation function to the data, executing it in a secure sandboxed environment and returning the results as a response (see Figure 7).

Figure 7 Response from the Graftwerk service

Previews, which are ultimately meant for display in a UI context will be returned in a slightly different format to the data transformation itself. This is especially important for Grafts, as they need to locate the position of a linked data fragments or errors in the graph template.

4 Conclusion

The Grafter data transformation framework has been built to support the DaPaaS methodology, enabling a wide range of data sources to be transformed efficiently into Linked Data.

The Graftwerk system has been built to allow Grafter to be used as a service, supporting integration of Grafter data transformation pipelines into the rest of the Datagraft platform.