DataGraft: One-Stop-Shop for Open Data Management[1]

Editor(s): Name Surname, University, Country
Solicited review(s): Name Surname, University, Country
Open review(s): Name Surname, University, Country

Dumitru Roman[a,*], Nikolay Nikolov[a], Antoine Putlier[a], Dina Sukhobok[a], Brian Elvesæter[a], Arne Berre[a], Xianglin Ye[a], Marin Dimitrov[b], Alex Simov[b], Momchill Zarev[c], Rick Moynihan[d], Bill Roberts[d], Ivan Berlocher[e], Seonho Kim[e], Tony Lee[e], Amanda Smith[f], and Tom Heath[f]
[a]*SINTEF, Forskningsveien 1a, 0373 Oslo, Norway*
[b]*Ontotext AD, Tsarigradsko Shosse, 1784 Sofia, Bulgaria*
[c]*Sirma Mobile, Bulgaria*
[d]*Swirrl IT LTD, Springbank Road, Macfarlane Gray House, Stirling, FK7 7WT, Stirlingshire, United Kingdom*
[e]*Saltlux Inc., 6F Deok-il Building,967 Daechi-Dong, Guaugnam-Gu, 135-848 Seoul, Republic of Korea*
[f]*Open Data Institute, St. James Square, St. James House, GL50 3PR Cheltenham, United Kingdom*

Abstract. This paper introduces DataGraft (https://datagraft.net/) – a cloud-based platform for data transformation and publishing. DataGraft was developed to provide better and easier to use tools for data workers and developers (e.g. open data publishers, linked data developers, data scientists) who consider existing approaches to data transformation, hosting, and access too costly and technically complex. DataGraft offers an integrated, flexible, and reliable cloud-based solution for hosted open data management. Key features include flexible management of data transformations (e.g. interactive creation, execution, sharing, reuse) and reliable data hosting services. This paper provides an overview of DataGraft focusing on the rationale, key features and components, and evaluation.

Keywords: data transformation, data publication, data hosting, data-as-a-service, open data, linked data

1. Introduction and Motivation

For the past five years, government and non-government institutions in the EU and around the globe have increasingly made data accessible under open licenses and often in reusable formats [1]. Recent statistics clearly display the achievements of these efforts. According to the EU-funded project OpenDataMonitor, 28 European countries have published more than 237,500 datasets through more than 160 catalogues [2]. Compared to the Zettabytes of data that the Internet is estimated to host, 1.25 Terabyte of open data spread across Europe may still appear like a modest result. But given the limitations and difficulties which data publishers and consumers face so far, it is not. Further removing barriers from open data publication and consumption processes remains a primary concern.

DataGraft started with the mission to alleviate some of these obstacles through new tools and approaches that support a faster and lower-cost publication and reuse of open data.

[1] DataGraft is accessible at https://datagraft.net/. This paper presents the capabilities of DataGraft as of January 2016.

[*]Corresponding author. E-mail: dumitru.roman@sintef.no.

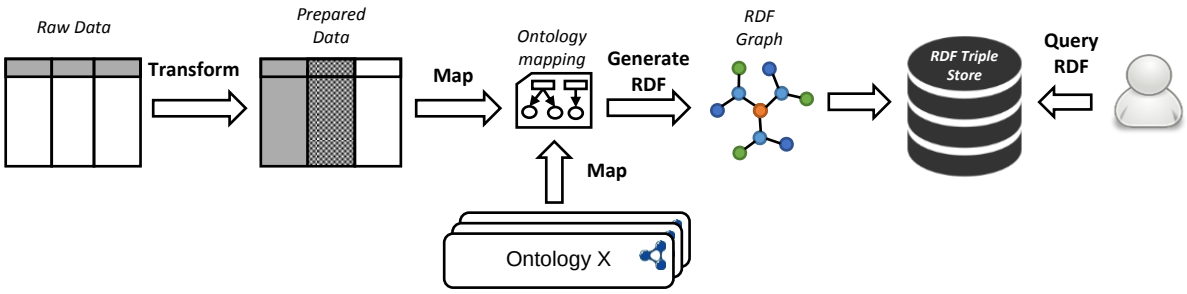

Figure 1. Typical data transformation/publication and access process: from tabular data to a queriable semantic graph

Open data is increasingly showing effects in solving problems for the public and private sectors, as well as addressing issues related to the environment and society. Figure 1 displays a typical process for the creation and provisioning of open linked data: *raw data* (most often tabular data) need to be cleaned, transformed, and prepared. Once data are prepared, they are mapped against a standard linked data *ontology* (which represents a data model) in order to then generate a semantic *RDF graph*. The resulting RDF is then stored in a semantic graph database, or *triple store*, where data users can easily access and query the data. Conceptually, this process is rather straightforward, however, to date, such an integrated workflow is not commonly implemented. Instead, publishing and consuming (linked) open data remains an intricate, tedious task due to a combination of the following three reasons:

1. The technical complexity of preparing open data for publication is high; toolkits are poorly integrated and require expert knowledge, particularly for more advanced publications of linked data that need to include consistent, manually curated metadata.
2. Even when the data preparation process is supported, organisations still face considerable costs to expose their data and provide reliable and scalable access to them. Especially in the absence of direct monetisation or other cost recovery incentives, the relative investment costs can easily become excessively high for many organisations. This might result in open data publishing initiatives being postponed, or executed in a way that makes data access and reuse difficult.
3. A poorly maintained and fragmented supply of open data also causes problems for those who want to reuse this resource. Firstly, in many cases, datasets are provided through a number of disconnected outlets. Additionally, even sequential releases of the same dataset are often formatted and structured in different ways. For example, column orders might have changed from one release to the next. Such basic errors make even very simple projects hard to sustain, e.g. running a web application, which relies on a single, continuously updated dataset.

Furthermore, there are a number of interesting problems that require smart solutions in order to assist data publishers and developers in the process depicted in Figure 1:

- *Interactive design of data transformations*: Designing transformations that provide instant feedback to users on how data changes can not only speed-up the process, but also provide users with mechanisms to ensure that the individual transformation steps result in the desired outcome.
- *Repeatable data transformations*: Very often a data transformation/publication process needs to be repeated as new data arrives (e.g., monthly budget reports are published through the exact same process each month). Executable and repeatable transformations are a key requirement for a scalable and lower-cost data publication process
- *Reusable and shareable data transformations*: Capabilities to reuse and extend existing data transformations created and shared by other developers further improves the speed and lowers the cost of the data publication process.
- *Distributed deployment of data transformations*: transforming data necessitates a varying resource utilization due to the varied load requirements with different inputs. Thus, having mechanisms to dynamically deploy and execute transformations in a distributed environment results in a more reliable, scalable and faster data publication process.
- *Reliable data access*: Once data are generated following a data transformation process, provisioning it reliably is another key aspect to ensure access to the data from 3rd party applications and services.

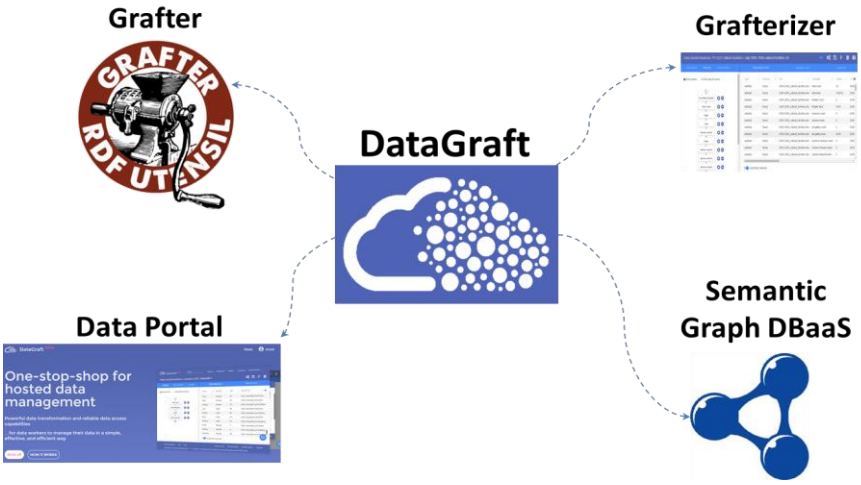

Figure 2. Core components of DataGraft

What is therefore needed is an integrated solution that enables a *self-serviced* effective and efficient data transformation/publication and access process. At the very core, this means automating the open data publication process to a significant extent - in order to increase the speed and lower its cost. What this will eventually lead to is that both data publishers and data consumers can focus on their goals:

- *Data consumers* can focus on utilising open data for data-driven decision making, or on creating new applications and services (rather than being data "hunters" and "gatherers")
- *Data publishers* can focus on providing high quality data in a timely manner, and finding monetization channels for their data (rather than spending time and resources on developing their own data publication & hosting platforms).

DataGraft was developed as a cloud-based platform for data workers to manage their data in a simple, effective, and efficient way, supporting the data transformation/publication and access process discussed above, through powerful data transformation and reliable data access capabilities.

The remainder of this paper provides and overview of DataGraft key features and core components (Section 2), evaluation (Section 3), discussion on related systems (Section 4), ending with a summary and outlook (Section 5).

2. DataGraft: Key Features and Components

DataGraft was designed and developed to support two core capabilities: data transformations and reliable data access.

For *data transformations*, DataGraft provides the following features:

1. Interactively build data transformations;
2. Deploy executable transformations to repeatedly clean and transform spreadsheet data;
3. Share transformations publicly;
4. Fork, reuse and extend transformations built by 3rd parties from DataGraft's transformations catalogue;
5. Programmatically access transformations and the transformation catalogue.

Related to *reliable data access*, DataGraft provides the following features:

1. Data hosting on DataGraft's reliable, cloud-based semantic graph database;
2. Query data through generated SPARQL endpoints or access it via linked data APIs;
3. Share data publically;
4. Programmatically access the data catalogue;
5. Operations and maintenance performed on behalf of users.

DataGraft realizes these capabilities through four core technical components, as shown in Figure 2:

Grafter is a software library for data cleaning and transformation. This is supplemented by **Grafterizer**,

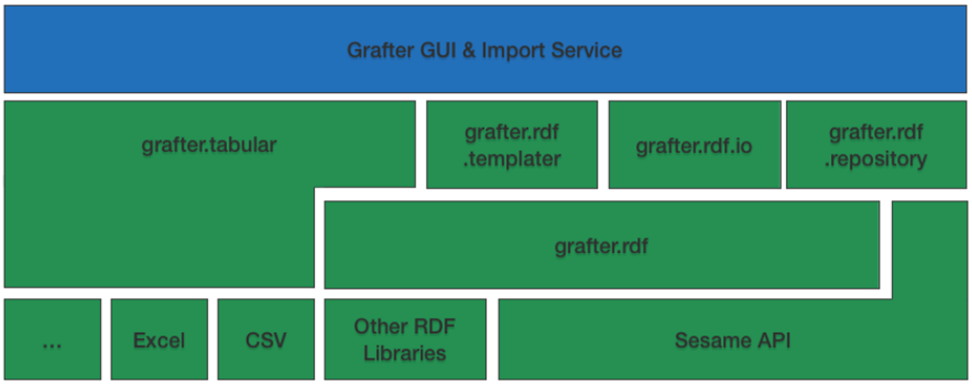

Figure 3. Grafter Architecture Stack

the front-end framework and interface for the underlying Grafter library. The **Semantic Graph Database-as-a-Service (DBaaS)**, establishes DataGraft's data warehouse for RDF data. Finally, the **DataGraft portal** ties together these service offers through a user-friendly, one-stop front-end. In the following we describe these components in further details and end with a summary of additional backend services and APIs.

2.1. Grafter

Grafter[2] is a service and a library of reusable components designed to support complex and reliable data transformations. At the heart of Grafter is a domain-specific language (DSL), which allows the specification of transformation pipelines that convert tabular data (e.g., for the purpose of cleaning it up), or produce linked data graphs.

It is implemented using Clojure, 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 implementation it becomes straightforward to exploit the large collection of useful libraries already available for the ecosystem of JVM-based programming languages.

Grafter benefits additionally include:

- Transformations implemented as pure functions on immutable data, which makes the logic of the transformation process significantly easier to reason about.
- Grafter supports large data transformations efficiently. Unlike other tools it takes a streaming approach to processing data, which means the maximum size of dataset that can be processed is not limited by the 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 semantic graph databases (triple stores) via standard interfaces.
- It has a highly modular and extensible design.

Grafter is composed of a number of modules 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`.

These two primary divisions represent the two sides of the Extract-Transform-Load (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).

[2] http://grafter.org/

```
(-> (read-dataset data-file)
    (make-dataset move-first-row-to-header)
    (rename-columns (comp keyword slugify))
    (melt [:area_id :iz-name :period])
    (derive-column :area-uri [:area_id] statistical-geography)
    (derive-column :variable-slug [:variable] name)
    (derive-column :alc-hosp-slug [:variable-slug :period :area_id] slug-combine)
    (derive-column :alc-hosp-uri [:alc-hosp-slug] alc-hosp-id)
    (derive-column :alc-hosp-var [:variable-slug] alc-hosp-def)
    (derive-column :label [:variable] (comp alc-hosp-cs-label name))))
```

(a)

```
(def alcohol-hospital-admissions-template
  "RDF template for Glasgow dataset 'alc_hosp_admissions_08.csv'"
  (graph-fn [{:keys [alc-hosp-uri alc-hosp-var area-uri label iz-name period variable value]}]

            (graph (base-graph "health/alcohol-hospital-admission")
                   [alc-hosp-uri
                    [rdf:a qb:Observation]
                    [rdfs:label (rdfstr (str label ", " period ", " iz-name))]
                    [qb:dataSet alc-hosp-ds]
                    [alc-hosp-var (parseValue value)]
                    [refPeriod (year-prefix period)]
                    [refArea area-uri]])))
```

(b)

Figure 4. Grafter sample code of data transformation (a) and creation of RDF triples (b).

The `grafter.tabular` namespace contains a wide variety of data processing functions for filtering data (by row, column or cell contents) and applying user-specified transformation functions to cells through functions like `derive-column`. Additionally, it includes more complex and powerful functions for normalising data into a more uniform form 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.

Tabular Grafter transformations are typically expressed as a sequence of step-wise operations on whole tables of data. All tabular operations are simply pure functions that take a *dataset* (a table) as input, and produce a *dataset* as output.

This can be seen in the example Grafter code in Figure 4 (a). This tabular dataset transformation processes a spreadsheet containing all alcohol-related hospital admissions in Glasgow. Each line represents a function call that receives a *dataset* (table) and returns a new one that has been transformed. Sequences of tabular operations such as those, where a table is received as input and returned as output, are called *pipes*.

Pipes are simply a set of 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. The interesting property about *pipes* is that they can be composed together arbitrarily, and always result in a valid *pipe*. Additionally because the inputs, outputs and intermediate steps to pipes are always tables; they are very intuitive for users to manipulate and use.

In order to publish linked data, a final transformation must take place to produce the graph structure used by RDF. To achieve this, a final step, referred to as creating a *graft*, is required. This steps 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.

Because a *graft* takes as input a table and returns a lazy sequence of quads representing the linked data graph as its output, it doesn'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 is 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 sub-DSL to specify commonly-used data

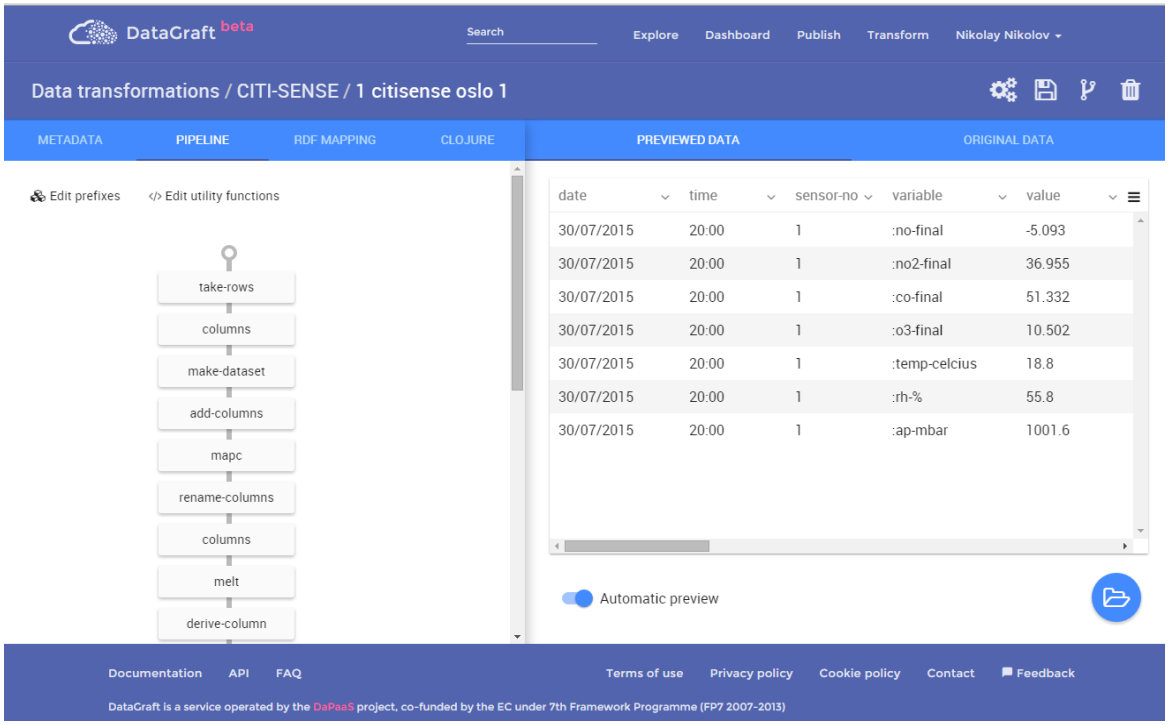

Figure 5. Grafterizer depicting a transformation pipeline (left) and intreeactive preview of data (right)

transformation functions, combined with selections from the input data and literal values.

The code in Figure 4 (b) is used to express the mapping of columns in a tabular *dataset* into its position in a linked data graph.

In DataGraft, Grafter comes together with **Graftwerk** – a back-end service (accessed through a RESTful API) for executing transformations. Graftwerk provides a sandboxed Platform-as-a-Service (PaaS) execution environment for Grafter transformations and supports two primary platform features:

1. The ability to execute a given 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 providing a high quality interactive user experience via the interface. Graftwerk supports both of these features on both kinds of transformations: *pipes* and *grafts*.

Further information about Grafter and Graftwerk can be found in [3,4].

2.2. Grafterizer

Grafterizer is the web-based framework for data cleaning and transformations based on Grafter. It provides an interactive user interface with a wide array of functionalities (screenshot in Figure 5). Together, these features provide end-to-end support for the data cleaning and transformation process:

- *Live preview* – Grafterizer interactively displays the results of the tabular clean-up or transformation steps in a side-panel. It also retains a view of the original version of the uploaded tabular dataset. Additionally, in case errors in the transformation or RDF mapping are present, it is equipped with an integrated error reporting capability.
- *Forking of existing transformations* – the user interface allows users to create copies of transformations by a single click of a button.
- *Specifying and editing data cleaning (pipeline) steps* – the clean-ups and transformations performed on tabular data can be added, edited, reordered or deleted. All functions are parameterised and editing allows users to change each of these parameters within the function with immediate feedback.

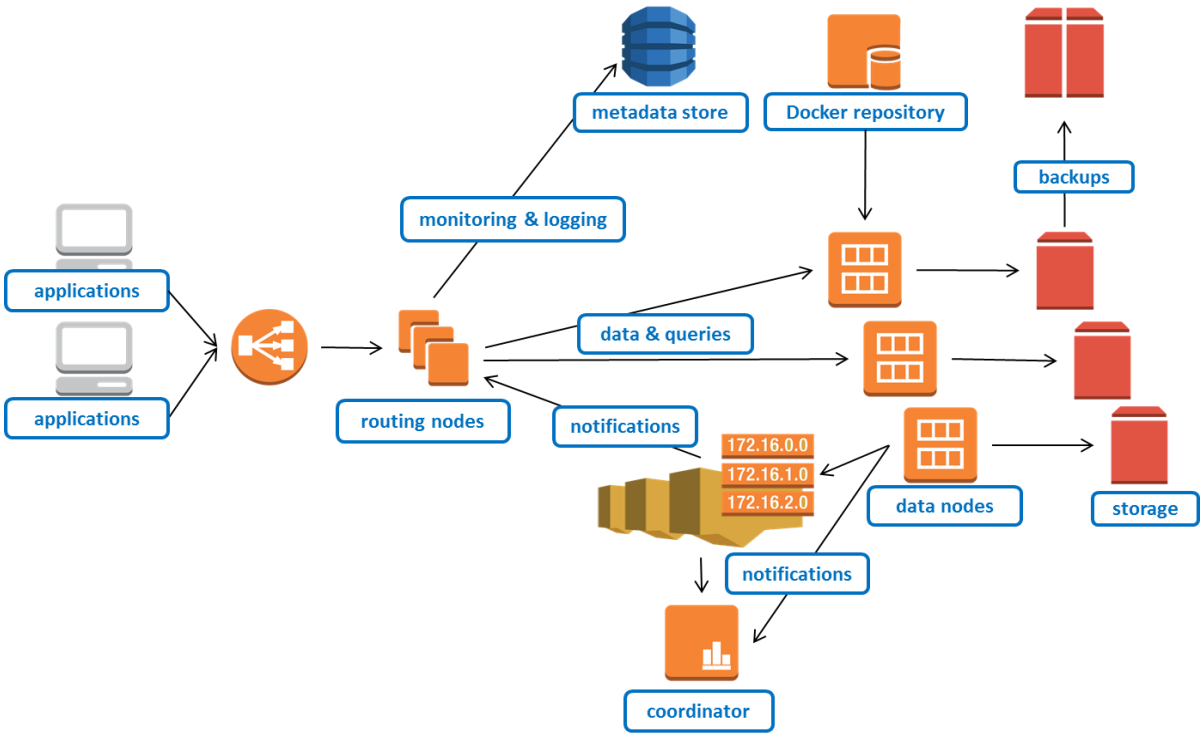

Figure 6. Architecture of the RDF DBaaS

- *Data page generation* – based on the specified RDF mappings, users are able to directly produce and publish data pages where their data will be available for access through an endpoint.
- *Direct download of resulting data* – the cleaned-up/transformed data from Grafterizer (both CSV and mapped RDF) can be directly accessed and downloaded locally.
- *Customisation* – data clean-up and transformation are fully customisable through embedding custom code, both as individual clean-up steps, or part of certain steps. In addition, developers can directly edit the resulting Clojure code and see the result in interactive mode.

Grafterizer implements a web-based wrapper over the Grafter library and Graftwerk service. The interface allows users to specify Grafter transformations in a much easier and more intuitive manner, compared to directly coding in Clojure. It also provides instant feedback alongside the other features described in the previous section.

The Grafterizer interface works by submitting the transformation that is being specified to the Graftwerk service along with the data that needs to be transformed. Depending on the type of the request, Graftwerk will then either generate a preview of the transformed data that the UI can display, or return the transformed data.

The result of a transformation can be either a set of RDF triples that can hosted on DataGraft in its semantic graph (RDF) store (described in the next subsection), or a tabular dataset that can be downloaded or accessed from the platform's file storage.

Further information about Grafterizer and its integration with DataGraft can be found in [5].

2.3. Semantic Graph Database-as-a-Service

DataGraft's database-as-a-service is a fully managed, cloud-based version of GraphDB™ semantic graph database (triple store), which provides an enterprise-grade RDF database as-a-service. Users therefore do not have to deal with typical administrative tasks such as installation and upgrades, provisioning and deployment of hardware, back-up and restore procedures, etc. The utilization of cloud re-

sources by the DBaaS depends on the load of the system itself, whereby they can be elastically provisioned and released to match the current usage load.

From a user standpoint, the DBaaS supports an API for linked data access, querying, and management. These functionalities are based on a complex architecture, which ensures components scalability, extensibility and availability on large scale (see Figure 6).

The DBaaS implementation follows the principles of micro-service architectures, i.e. it is composed of a number of relatively small and independent components. The data management architecture is based on the Amazon Web Services (AWS) cloud platform is comprised of the following components:

- *Load balancer* – the entry point to the database services is the load balancer provided by the AWS platform, which routes incoming data requests to one of the available routing nodes. It can distribute requests even between instances in different datacentres.
- *Routing nodes* host various micro-services such as: user authentication, access control, usage, metering, and quota enforcement for the RDF database-as-service layer. The front-end layer is automatically scaled up or down (new instances added or removed) based on the current system load.
- *Data nodes* – this layer contains nodes running multiple instances of the GraphDB™ database (packaged as Docker[3] containers). Each data publisher has their own database instance (container), which cannot interfere with the database instance or with the data of the other users of the platform. The data are hosted on network-attached storage volumes (EBS)[4] and each user/database has their own private EBS volume. Additional OS-level security ensures appropriate data isolation and access control.
- *Integration services* – a distributed queue and push messaging service enable loose coupling between the various front-end and database nodes on the platform. All components use a publish-subscribe communication model to be aware of the current state of the system. This allows the front-end and the backend layers to be scaled up or down independently as they are not aware of their size and topology.
- *Distributed storage* – all user data are stored on the network-attached storage (EBS), whereas static backups and exports remain on the S3 distributed storage. Logging data, user data as well as various configuration metadata are stored in a distributed NoSQL database (DynamoDB).
- *Monitoring services* – the AWS cloud provides various metrics for monitoring the service performance. The DBaaS utilises these metrics in order to provide optimal performance and scalability of the platform. The different layers of the platform can be automatically scaled up (to increase system performance) or down (to decrease operational costs) in response to the current system load and utilisation.

Further information about semantic graph database can be found in [6].

2.4. DataGraft Portal

The DataGraft portal integrates the previously discussed components together in a modern, user-friendly web-based interface designed to ensure a natural flow of the supported data processing and publication workflow. Accordingly, four main aspects have been considered and addressed throughout the development of DataGraft's portal front-end:

1. Design and implement a highly intuitive UI, which facilitates user interaction with large, complex sets of linked open data.
2. Simplify the data publishing process by implementing a fast-track for publishing data, e.g., using simple drag and drop operations.
3. Create basic data exploration tools, which help users with limited technical skills to explore data hosted on the DataGraft platform.
4. Deploy data visualization components that can be easily used by non-specialists to build data driven portals.

Two complementary modules have been implemented to create the DataGraft portal. First, a drag-and-drop interface, which allows a user to easily publish and annotate data. The entire process of publishing data is thus reduced to a simple wizard-like inter-

[3] https://www.docker.com/
[4] http://aws.amazon.com/de/ebs/

face, where publishers can simply drop their data and enter some basic metadata. Secondly, the DataGraft portal provides a module that helps visualize data from the semantic graph database (triple store). Publishing data on the web usually implies specific programming skills in order to create web pages or portals. Here, the programming process has been all but eliminated through the deployment of visualization widgets that can serve as reusable components for data publishing. These widgets can access and use data in the repository and expose it through a data page. Currently, the platform provides a number of visualization widgets, including tables, line charts, bar charts, pie charts, scatter charts, bubble charts and maps (using the Google Maps widget). All widgets are populated with data through the use of specific SPARQL queries on RDF databases.

Figure 7 depicts the configuration of a line chart widget using SPARQL (top part of the picture) for comparing statistical data on employment in two municipalities in Norway, and the result of the visualization on a data page (lower part of the widget).

Further information about the DataGraft portal can be found in [5,7].

Figure 7. Configuring widgets (top) and visualizing data (down) in DataGraft

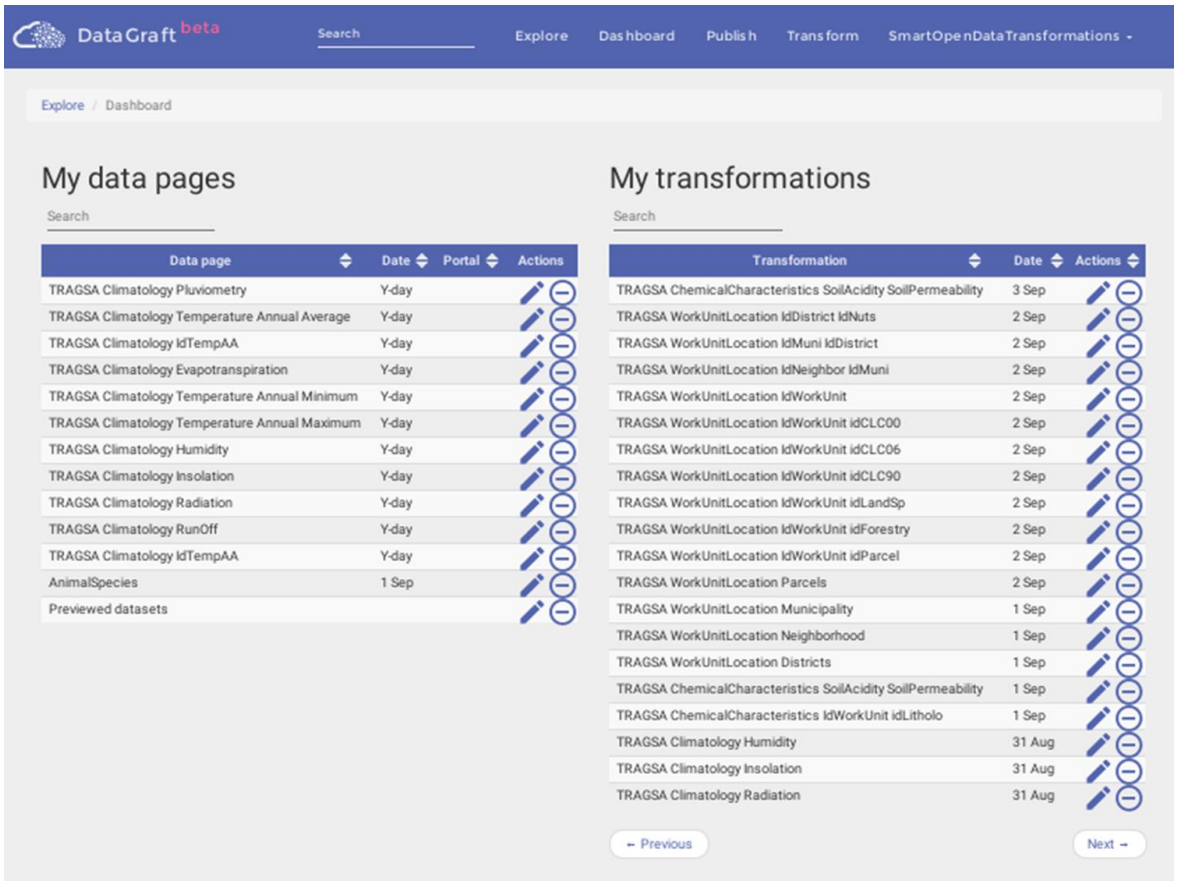

Figure 8. Data transformation catalogue (right) and data pages catalogue (left)

2.5. DataGraft Backend Services and APIs

In addition to the core components described above, DataGraft comes with a set of services (that can be accessed through RESTful APIs) for managing data and transformations, user management, security and authentication.

DataGraft provides capabilities to search the data pages or data transformation catalogues. Figure 8 depicts the data transformations and data pages catalogues as seen by a user on DataGraft.

Examples of back-end services for data transformations include CRUD operations on transformations catalogue (create, retrieve, update, or remove data transformations), as well as distribution of transformations code and services supporting the interactive preview of transformations.

Examples of back-end services for data include: CRUD operations on data pages catalogue (create, retrieve, update, or remove data pages); distribution of data (DCAT[5] compliant); services for managing DBaaS instances; and services for querying of the linked data graphs using the OpenRDF API.

Further information about the DataGraft backend services can be found in [5,6].

Table 1 summarizes the technical components of DataGraft and Figure 9 provides a high level components interactions.

Table 1. Summary of DataGraft components and their capabilities

Grafter	Clojure-based DSL for data transformations
Graftwerk	Grafter/Clojure execution engine
Grafterizer	Front-end framework for data cleaning and transformations
Portal UI	Dashboard, exploration, user data, API keys, upload of files, data pages, transformation pages

[5] http://www.w3.org/TR/vocab-dcat/

Back-end services	User management, cataloguing, transformation management, data publishing
Semantic graph DBaaS	Cloud-based data-as-a-service component

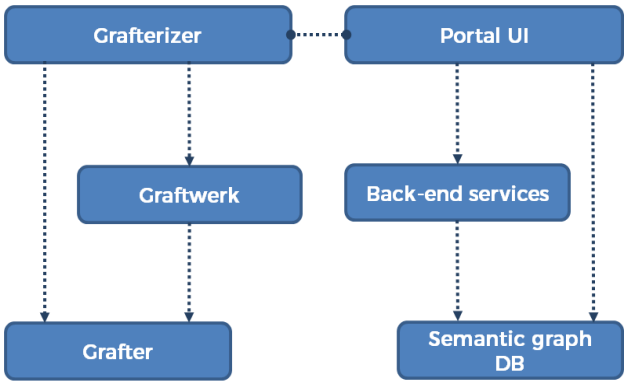

Figure 9. DataGraft high level components interactions

3. Evaluation

The DataGraft platform was launched in public beta in September 2015. Early usage figures show that the platform, by beginning of December 2015, has:

- 495 (91 public) registered data transformations
- 1520 uploaded files
- 181 registered users
- 183 public data pages

DataGraft has been used in various domains to transform and publish data. In general, the following positive aspects resulted from the use of DataGraft in practice:

- Simplified data publishing process
- Time-efficient data transformation and publishing process
- Repeatable and sharable data transformation process
- Data hosting and querying support, with possibility to visualize data
- Support for integration of transformed data with external data sources using established web standards

In the following we provide some examples of how DataGraft was used in practice, the positive aspects and limitations identified in these examples.

3.1. PLUQI

Saltlux,[6] a South Korean company operating in the domain of knowledge management, developed a Web application called the *Personalised Localised Urban Quality Index* (PLUQI). PLUQI implements a customizable index model that can be used to represent and visualize the level of well-being and sustainability for given cities based on individual preferences.

Aside from building an attractive and engaging end-user interface, the main challenges associated with PLUQI are to integrate and/or merge data from various sources, e.g., open data portals, social sensor systems, etc. This index model takes into account data from various domains such as daily life satisfaction (weather, transportation, community, etc.), healthcare level (number of doctors, hospitals, suicide statistics, etc.), safety and security (number of police stations, fire stations, crimes per capita, etc.), financial satisfaction (prices, income, housing, savings, debt, insurance, pension, etc.), level of opportunity (jobs, unemployment, education, re-education, economic dynamics, etc.), environmental needs and efficiency (green space, air quality, etc.).

PLUQI was implemented in two iterative versions. The first implementation used Korean data, providing a semantic integration of heterogeneous open data from the Korean Statistical Information Service (KOSIS). The second implementation has been developed based on similar data about Scotland (from the statistics office of the Scottish Government, and other open data), and it compares the PLUQI index between Edinburgh and other council areas.

A typical process for implementing an application such as PLUQI requires four main steps: (1) data gathering (identifying relevant data sources), (2) data transformation (cleaning, harmonization, integration), (3) data provisioning (making integrated data reliably available), and finally (4) implementing the application (e.g. Web application, visualizing and accessing the provisioned data). DataGraft was used for steps (2) and (3) for transforming/integrating data, and reliably provisioning the data. Saltlux reported reduction of cost for implementing these steps of approximately 23% compared to traditional approaches for integrating and provisioning data. This enabled Saltlux to

[6] http://saltlux.com/

focus on steps (1) and (4) while outsourcing the other two steps to DataGraft.

DataGraft was perceived as a convenient platform to integrate various datasets into one repository based on the linked data approach. It supports transformation features for raw datasets, which are not written in RDF format so that they can be retrieved using SPARQL queries, linking them to other datasets. This functionality was recognised as very powerful for services such as PLUQI that need to use various datasets. Data are also expected to be retrievable from one repository, enabling a reasonable response time against the user's requests. PLUQI illustrates the capability of DataGraft to support a wide variety of data integration and analysis applications.

The capability of DataGraft to allow users to share data transformations was considered as a really useful feature when transforming datasets with related structure. For example, when transforming air quality data for different cities – one transformation pipeline for one city was reused with small modifications to transform air quality data from other cities.

When it comes to limitations, PLUQI revealed the need for DataGraft to export or share visualization widgets that can be used by third party services or applications: currently, the data pages in DataGraft provide capabilities to visualize datasets in various widgets, but it was pointed out by PLUQI developers that it would be useful to implement exporting and sharing features for widgets, in order to raise the usability of the portal. In addition, DataGraft did not provide capabilities for ontology editing, and therefore the PLUQI developers had to use third-party ontology editing tools such as Protégé to create the ontologies against which that data was annotated in the RDF publication process.

Further information about the PLUQI application can be found in [8,9].

3.2. Transforming and Publishing Environmental Data

As part of the SmartOpenData project[7], DataGraft was used to transform and publish data as open data in the biodiversity and environment protection domain from two organizations: TRAGSA, a Spanish company, and ARPA, the environmental protection in Italy. In the case of TRAGSA, 42 transformations were created, out of which 25 were created via the reuse/forking capability of Grafterizer. For the purposes of ARPA, five transformations were created, two out of which were created via the reuse/forking capability. The following aspects were noted during the data transformation/publication process:

- The ability to 'fork' (copy, then adapt) existing transformations allowed users to easily reuse existing transformations and thus save time in the process of creating a new transformation
- The ability to edit the parameters of each transformation step interactively, and to change the order of steps, helped them to: create transformations in general, detect and correct mistakes, experiment with different parameters for transformation steps, specify proprietary transformation designs (through the ability to add utility functions with custom code) and reuse proprietary functions across different transformations.

The users identified some features not currently available in Grafterizer that would have made the tools more useful for them, notably the ability to join more than one input dataset and the ability to sort datasets. To overcome these limitations, it was necessary to carry out some pre-processing of the input files (e.g., for one transformation, 27 of the 43 files tested required some pre-processing). It is possible to perform joins and to sort inside a Grafter pipeline, but at the time of the writing of this paper, the Graftwerk back-end did not yet provide explicit support for these operations.

3.3. Other Examples

For example, Statsbygg[8], the Norwegian government's key advisor in construction and property affairs, has experimented with DataGraft for the purpose of publishing their data about public buildings in Norway and integrating it with external information about accessibility in buildings (e.g. types of facilities for the disabled). This was done in order to provide better information about public buildings in Norway, and have a more efficient mechanism to share data about public buildings. In this context, DataGraft was used for a wide range of tasks, including cleaning of tabular data about buildings, data transformation, generation and hosting of RDF data, integration with external data sources, querying the integrated data (e.g. which public buildings are located in Oslo and

[7] http://www.smartopendata.eu/

[8] http://www.statsbygg.no/

don't have handicapped entrances), and visualization of public buildings and associated data on a map.

Noted benefits for use of DataGraft in this context included the possibility to create a "live" services (vs. storing and managing information about public buildings using spreadsheets) that can be easily updated as new data becomes available, efficient sharing of data, and simplified integration with external datasets. On the other hand, lack of selective data sharing in DataGraft was pointed out – it is often the case that some buildings such as prisons or King's properties are not to be shared in an open fashion. This is not necessarily a technical deficiency of DataGraft but it's rather expensive to enforce it in practice. Group- or role-based access control of DataGraft assets (which as of the writing of the paper is not supported in DataGraft), could be a potential solution to share data selectively.

Another example where DataGraft has been applied is in publishing statistical data as RDF, querying and visualization. For instance, DataGraft was applied for cleaning and transforming statistical data from StatBank Norway[9]. StatBank contains detailed tables with various time series data. Users can create custom selections and serialise them in different file formats. An interesting observation was the ease of use of repeatable and reusable transformations provided by DataGraft, which allowed users to repeatedly apply and adjust transformations on the tables generated from StatBank Norway. Figure 7 depicts an example of configuring a widget and visualizing the RDF data transformed from StatBank Norway. In this context, users pointed out the limitation of DataGraft of currently allowing only CSV files as input for data transformations. The limitation is related to the capabilities of Graftwerk (the transformation execution service), and not of Grafter. As a matter of fact, Grafter, as a standalone library, has been applied by Swirrl[10], a UK SME working in data publishing, to generate a large number of linked data datasets for local government organisations in the UK (including Glasgow, Hampshire, Surrey and Manchester councils) whose source data covered a range of file formats, including CSV, Excel, PostGIS (relational database with GIS extensions), and ESRI Shapefiles (a common GIS format).

DataGraft has been used in other contexts for transforming and publishing data, such as air quality sensor data from the CITI-SENSE project[11], data about transportation infrastructure (e.g. the road network, bridges, etc.) and the impact of natural hazards in transportation infrastructure as part of the Infra-Risk project[12], or investigating crime data in small geographies[13].

Currently DataGraft is offered as a completely free service with some imposed limitations per account as follows:

- Data upload: Users can upload CSV files of up to 10MB each, and RDF files of up to 100MB each;
- Data pages: Users can have up to 10 RDF data pages;
- Persistent storage: Users can host up to 2 GB of CSV data, and 1 Million RDF triples for RDF data.

As of now, these limitations are enforced in order to maintain DataGraft as a free offering to its users. Nevertheless, DataGraft does allow for the transformation of much larger quantities of data. This is implemented through taking advantage of the Clojure (Grafter) compatibility with the JVM. This means that all transformations specified in DataGraft can be bundled into executable Java Archives (JAR files). These JARs are available through a DataGraft service and can be downloaded and executed locally to process arbitrarily large inputs.

4. Related Systems

DataGraft offers an integrated, flexible, and reliable cloud-based solution for data transformation and data access. Its uniqueness stems from its ability to seamlessly integrate useful data transformation capabilities (e.g. interactive creation, execution, sharing, and reuse of data transformations) with reliable hosting/access of data, with a particular focus on the linked data paradigm. A notable differentiation of DataGraft is the fact that it alleviates users from the complexity of provisioning resources and managing the various capabilities for transforming and hosting data – which is not the case with most of the related approaches. Typically, they provide interesting tools but lay the burden of provisioning of resources and maintenance on the users.

[9] https://www.ssb.no/en/statistikkbanken
[10] http://www.swirrl.com/
[11] http://www.citi-sense.eu/
[12] http://www.infrarisk-fp7.eu/
[13] http://benproctor.co.uk/investigating-crime-data-at-small-geographies/

Related systems fall under the types of systems for linked data and open data publication/hosting and ETL tools. In the following we outline the most relevant systems and discuss the differences.

The Linked Data Stack[10] is a software stack consisting of a number of loosely coupled tools, each capable of performing certain sets of operations on linked data, such as data extraction, storage, querying, linking, classification, search, etc. The various tools are bundled in a Debian package and a web application can be deployed to offer a central access to all the tools via a web interface[14]. The complexity of provisioning resources and managing the resulting web application rests on end users who must install the tools and maintain the infrastructure. In contrast, DataGraft is bundled as-a-service, whereby the actual resources for transforming and hosting data are managed on behalf of the user. Whereas there is indeed an overlap in terms of operations on linked data (for example data hosting and querying) supported by both DataGraft and the Linked Data Stack, there are also significant differences in approach. Whereas DataGraft focuses on higher level aspects such as providing an integrated framework for data transformation and cleaning such that users can interactively design and share transformations, the Linked Data Stack addresses lower level aspects, such as classifications, that are not covered in DataGraft.

The LinDA project [11] [15] developed a set of tools for linked data publishing, packaged into the LinDA Workbench[16]. It consists of a lightweight transformation to linked data tool, a vocabulary repository, a tool for converting RDF to conventional data structures, a visual query builder, and an analytics package. Similar to the Linked Data Stack, the tasks of provisioning resources and managing the LinDA tool ecosystem again rest on the end user. Furthermore, DataGraft's powerful data cleaning and transformation approach goes beyond the lightweight transformation tool provided by LinDA (which focuses primarily on the RDFisation). Nevertheless, LinDA provides more sophisticated support for visual querying through providing a query builder for SPARQL.

The COMSODE project [12][17] provided a set of software tools and methodology for open data processing and publishing. A relevant tool developed as part of this project was UnifiedViews[18] – an ETL tool for RDF data. Its focus is on specifying, monitoring and debugging workflows composed of data processing units applied on data that is extracted from SPARQL endpoints. These features are orthogonal to DataGraft's transformation approach that focuses on lower level operations such as cleaning and RDFization of the actual data (together with other aspects like sharing and reuse of transformations). Thus, UnifiedViews can be seen as a data workflow processing tool using data published via DataGraft. Similar to the aforementioned approaches, COSMODE is not available as an online service, but rather as a set of tools that need to be individually managed, which implies additional burden on end users.

OpenRefine[19], with its RDF Refine plugin [13][20], implements an approach with similar capabilities to DataGraft with regards to data cleaning, transformation, and mapping to RDF. The tool provides an interactive user interface that uses well-known spreadsheet-style interactions, which are convenient for manual data clean-up and conversion. However, OpenRefine is unsuitable for use in a service offering context, such as the one DataGraft was built for. Although OpenRefine was implemented as a web application, its design is monolithic and bears resemblance to an application meant for the desktop, rather than the web. Firstly, the code base is not well componentised – e.g., the transformation engine is tightly-coupled to the OpenRefine core and does not expose an API. Additionally, the processing engine itself is not suitable for robust ETL processes, as it is inefficient with larger data volumes – it implements a multi-pass approach to individual operations, and is thus memory-intensive. Although there has been an attempt to provide support for batch processing in BatchRefine[21], it inherits the issues with the tight coupling of the core components. OpenRefine also has security issues, which prevent it from being applicable in a fully hosted solution. Nevertheless, OpenRefine provides more powerful RDF mapping features such as automatic reconciliation of data, more freedom in mapping, etc.

PublishMyData[22] is a commercial Software-as-a-Service linked data publishing platform. DataGraft and PublishMyData share the data transformation approaches through the Grafter library. However, DataGraft has taken Grafter further in the Grafterizer tool and the platform UI, by complementing it with interactive design of transformations, mechanisms for

[14] http://demo.lod2.eu/lod2demo
[15] http://linda-project.eu/
[16] http://linda-project.eu/tools/
[17] http://www.comsode.eu/
[18] http://unifiedviews.eu/

[19] http://openrefine.org/
[20] http://refine.deri.ie/
[21] https://github.com/fusepoolP3/p3-batchrefine
[22] http://www.swirrl.com/publishmydata

sharing transformations between users, and reliable data hosting and access. DataGraft is meant to be a public freely available service offering, rather than a commercial solution.

The Linked Data AppStore [14] is a Software-as-a-Service platform prototype for data integration on the Web. It integrated a set of linked data related tools for tasks such as data cleaning, transformation, entity extraction, data visualization, crawling in a Software-as-a-Service prototype and served as inspiration for further work on DataGraft.

Related systems that are not focused on the linked data paradigm include solutions for open data catalogues and more traditional ETL tools. Open data catalogues typically list open datasets, together with the metadata and references (e.g. URLs) to where the data are made available. In some cases the actual datasets are also hosted with the data catalogue. However, in most cases data catalogues provide only download links without any sophisticated infrastructure for querying data to support other aspects of the ETL process. Popular data catalogue solutions include CKAN[23] and Socrata[24]. The core differences between DataGraft and these data catalogue solutions lay in the support for linked data and data transformations.

Commercial examples of relevant ETL tools include Pentaho Data Integration[25] and Trifacta Wrangler[26]. Pentaho Data Integration is a powerful and efficient tool designed specifically for ETL processes with lots of plugins and components for many data formats. It does not come with linked data support and is rather hard to use for non-developers. Firstly, it does not provide an interactive preview such as a spreadsheet of the current state of the transformation workflow. Furthermore, its diagrammatic approach to transformations can be unclear and could potentially explode in complexity including constructs for loops and recursion. Finally, Pentaho Data Integration was not built for public cloud hosting environments, but is rather a desktop application. Trifacta Wrangler is yet another tool for data cleaning and transformation. Its focus is on supporting predictive interactions which enables users to clean and transform data in a rather simple manner. However, Trifacta Wrangler does not support linked data and capabilities or data hosting.

[23] http://ckan.org/
[24] http://www.socrata.com/
[25] http://community.pentaho.com/projects/data-integration/
[26] https://www.trifacta.com/products/wrangler/

5. Summary and Outlook

DataGraft is an emerging solution (as-a-Service) for making open linked data more accessible. It comes with a platform, portal, methodology, and APIs – all packaged in an online service, functional and documented[27]. DataGraft has been validated in a number of use cases showing added-value based on its key capabilities: support for sharable, repeatable, and reusable data transformations, and reliable RDF Database-as-a-Service.

DataGraft's features include: interactive cleaning and transforming data, repeatable and reusable data transformations, flexible deployment of transformations, RDF data publication and querying, support for integrating and visualising data from different sources. Additionally, DataGraft can be used with 3rd party tools as it has been built with easy integration in mind using standard web technology. For example, users can browse data hosted on DataGraft with tools such as GraphRover[28] (through the SPARQL endpoint), connect to the hosted data store with a standard Sesame Client (using the OpenRDF APIs), or browse, perform queries or other actions using a REST client or command line tools such as httpi[29] (to access the RESTful APIs of DataGraft). This makes DataGraft attractive both for data workers and data developers interested in simplified and cost-effective solutions for managing their data. DataGraft was developed to provide better and easier to use tools for open data publishers, linked data developers, and data scientists who consider existing approaches to data transformation, hosting, and access too costly or technically complex.

Graftwerk and RDF DBaaS are closed source, while Grafter[30], Grafterizer[31], and the DaPaaS portal[32] are released under open-source (EPL v1.0).

DataGraft is currently under development and operated by the proDataMarket project[33]. Changes and improvements to DataGraft are expected in the near future. Future releases of DataGraft will address some of the limitations identified in the cases in which DataGraft was used, and will contain new features and improvements. Examples of new features/improvements in the future releases include:

[27] https://datagraft.net/documentation/ (API documentation available at https://datagraft.net/api/)
[28] http://www.metreeca.it/products/graph-rover/
[29] http://httpirb.com/
[30] https://github.com/Swirrl/grafter
[31] https://github.com/dapaas/grafterizer
[32] https://github.com/dapaas/datagraft
[33] http://prodatamarket.eu/

support for multiple files, joining of datasets and various data formats as input for data transformations (e.g., JSON, GML, Shapefile), better error reporting in data transformations, applying functions directly on the preview spreadsheet (rather than in pipeline operations), dealing with streams of data (rather than static files), better traceability for files, data pages, transformations; ability to store and share assets such as queries and visualization widgets; versioning of assets, social aspects (e.g., users following activity of other users).

Acknowledgements. The development of DataGraft was co-funded through grants from the European Commission (EC). DataGraft was developed and operated by the DaPaaS project (GA no. 610988) until November 2015. Further development, maintenance, and operations continue under the proDataMarket project (GA no. 644497). The projects SmartOpenData (603824) and InfraRisk (603960) also contributed to DataGraft.

References

[1] T. Davies, R. M Sharif, and J. M. Alonso. Open Data Barometer. Global Report - Second Edition, 2015. Available via http://bit.ly/1OedPRd (Last accessed December 2015).
[2] http://opendatamonitor.eu/frontend/web
[3] B. Roberts and R. Moynihan. Documented methodology and guidelines. DaPaaS Deliverable D4.1, October 2014. Available via http://bit.ly/1NMU8lJ (Last accessed December 2015).
[4] B. Roberts. Software tools integrated into platform. DaPaaS Deliverable D4.2, April 2015. Available via http://bit.ly/1PcM8v7 (Last accessed December 2015).
[5] A. Simov, M. Dimitrov, N. Nikolov, A. Pultier, D. Suhobok, X. Ye, D. Roman. Open Data PaaS prototype, v.2. DaPaaS Deliverable D2.3, July 2015. Available via http://bit.ly/1Jj86s4 (Last accessed December 2015).
[6] M. Dimitrov, A. Simov, N. Nikolov, D. Roman. Open DaaS prototype, v.2. DaPaaS Deliverable D1.3, July 2015. Available via http://bit.ly/1PgdhPU (Last accessed December 2015).
[7] M. Zarev. Cross platform data delivery framework. DaPaaS Deliverable D3.2, July 2014. Available via http://bit.ly/1PgdhPU (Last accessed December 2015).
[8] I. Berlocher, S. Kim, T. Lee. Use case implementation, v1. DaPaaS Deliverable D5.2, October 2014. Available via http://bit.ly/1Ib5uzJ (Last accessed December 2015).
[9] S. Kim, I. Berlocher, T. Lee. Use case final implementation and validation. DaPaaS Deliverable D5.3, October 2015. Available via http://bit.ly/1Nv3y5O (Last accessed December 2015).
[10] S. Auer et al. Managing the Life-Cycle of linked data with the LOD2 Stack. ISWC, Springer, 2012.
[11] P. Hasapis, et al. "Business value creation from linked data analytics: The LinDA approach." eChallenges e-2014, 2014 Conference. IEEE, 2014.
[12] P. Hanečák, S. Krchnavý, I. Hanzlík. “COMSODE publication platform – Open Data Node – final.” COMSODE Deliverable 4.3, July 2015. Available via http://bit.ly/1PcCGYS (Last accessed Dec 2015).
[13] F. Maali. "Getting to the Five-Star: From Raw Data to Linked Government Data." Master's thesis, National University of Ireland, Galway, Ireland (2011). Available via http://bit.ly/1TVLnaW (Last accessed Dec 2015)
[14] D. Roman, et al. "The linked data AppStore." Mining Intelligence and Knowledge Exploration. Springer International Publishing, 2014. 382-396.