

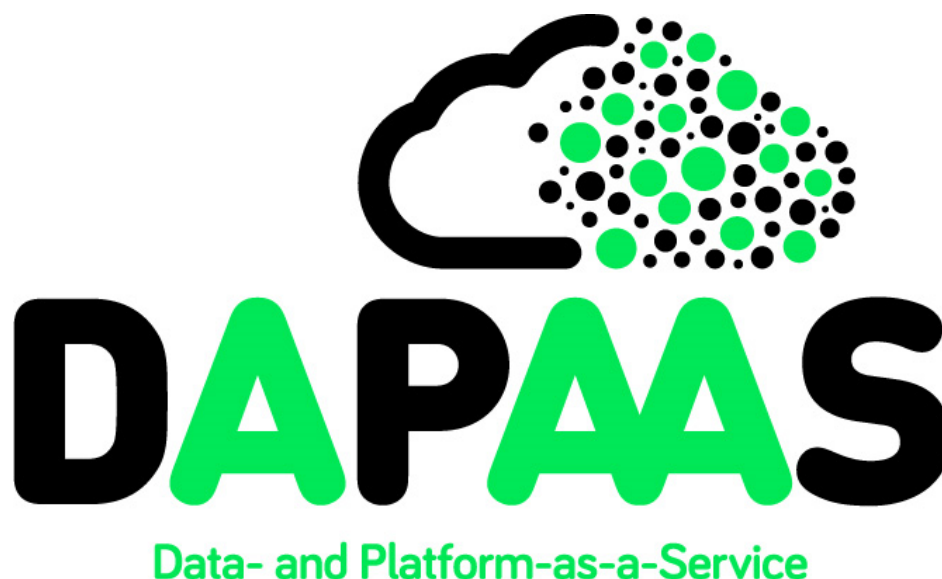
Small or medium-scale focused research project (STREP)

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 D2.2

Open Data PaaS prototype, v.1

| | |
|----------------------|---|
| Date: | 31.10.2014 |
| Author(s): | Brian Elvesæter (SINTEF), Dumitru Roman (SINTEF), Nikolay Nikolov (SINTEF), Alex Simov (Ontotext) and Marin Dimitrov (Ontotext) |
| Dissemination level: | PU |
| WP: | WP2 |
| Version: | 1.0 |

Document metadata

Quality assurors and contributors

| | |
|--------------------|--|
| Quality assuror(s) | Bill Roberts (Swirrl) and Seonho Kim (Saltlux) |
| Contributor(s) | DaPaaS Consortium |

Version history

| Version | Date | Description |
|---------|------------|--|
| 0.1 | 26.09.2014 | Initial outline and Table of Contents (TOC) |
| 0.2 | 15.10.2014 | Revised outline |
| 0.3 | 24.10.2014 | Description of the requirements addressed, data cleaning and app development |
| 0.4 | 27.10.2014 | User guide for Data Cleaning and first round of API documentation |
| 0.5 | 28.10.2014 | Ready for internal review |
| 0.6 | 31.10.2014 | Revision addressing reviewers' comments |
| 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.

This document describes the first version of the PaaS prototype developed for Deliverable D2.2 “Open Data PaaS Prototype, v.1” as defined in the Description of Work (DoW) of the DaPaaS project. The PaaS prototype consists of a set of software components that were developed based on the requirements, design and architecture specification from Deliverable D2.1. The development has been closely aligned with Deliverable D1.2 “Open DaaS prototype, v.1” in order to ensure seamless integration.

The software components developed for the first version of the PaaS prototype are:

- **User Management & Access Control**, which manages user profiles and secure access control to apps and datasets.
- **Data Cleaning & App Development**, which provides functionalities for applications development, and support for data cleaning & transformation and data workflows.
- **App Management & Deployment**, which gives developers control over the deployed applications and configuration settings for the application-hosting environment.
- **App Catalog** for searching and exploring apps and their metadata.

These components are available through the DaPaaS platform and accessible by data publishers, application developers and data consumers through REST APIs and graphical front-ends. User guides and API documentation are included in the Appendices of this document.

Table of Contents

| | |
|---|-----------|
| EXECUTIVE SUMMARY | 3 |
| TABLE OF CONTENTS..... | 4 |
| LIST OF ACRONYMS | 5 |
| LIST OF FIGURES | 6 |
| LIST OF TABLES | 7 |
| 1 INTRODUCTION | 8 |
| 2 REQUIREMENTS ADDRESSED BY THE PROTOTYPE..... | 9 |
| 3 DESIGN AND IMPLEMENTATION OF THE PROTOTYPE..... | 12 |
| 3.1 INTEGRATION WITH THE DATA LAYER | 12 |
| 3.2 USER MANAGEMENT & ACCESS CONTROL | 13 |
| 3.3 DATA CLEANING & APP DEVELOPMENT | 14 |
| 3.3.1 Data Cleaning & Transformation | 15 |
| 3.3.2 Data Workflows | 15 |
| 3.4 APPLICATIONS MANAGEMENT & DEPLOYMENT | 17 |
| 3.5 APPLICATIONS CATALOG | 17 |
| 4 DEPLOYMENT DETAILS | 20 |
| 5 FUTURE WORK..... | 21 |
| 5.1 DOCKER (EXTENDING THE APP HOSTING ENVIRONMENT)..... | 21 |
| 5.2 NOTIFICATIONS | 21 |
| 5.3 GRAFTER GUI ENHANCEMENTS (DATA CLEANING & WORKFLOWS) | 21 |
| 5.4 MANAGEMENT UI..... | 22 |
| 6 APPENDIX A: USER GUIDE FOR GRAFTER GUI | 23 |
| 7 APPENDIX B: API DOCUMENTATION | 28 |
| 7.1 USER MANAGEMENT API..... | 28 |
| 7.1.1 Results | 28 |
| 7.1.2 User Services & Web UI Authentication | 28 |
| 7.1.3 Account Service | 29 |
| 7.1.4 API Key Management..... | 32 |
| 7.2 DATA CLEANING & TRANSFORMATION API..... | 33 |
| 7.2.1 Transformation application API (Grafter Import) | 33 |
| 7.2.2 Transformations management API..... | 33 |
| 7.3 APPLICATIONS DEPLOYMENT API..... | 34 |
| 7.3.1 Deploy application API..... | 34 |
| 7.4 APPLICATIONS CATALOG API | 35 |
| 7.4.1 Catalog access API..... | 35 |
| 7.4.2 Applications descriptions access and management API..... | 36 |
| 7.4.3 Releases management API | 37 |
| 8 APPENDIX C: COMPARISON OF DATA CLEANING & TRANSFORMATION SOLUTIONS . | 39 |

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 |
| UML | Unified Modeling Language |

List of Figures

| | |
|--|----|
| Figure 1: Architecture of the PaaS layer..... | 12 |
| Figure 2: Architecture of the DaaS layer | 13 |
| Figure 3: Design of User Management & Access Control..... | 13 |
| Figure 4: Development scope for first version of the Data Cleaning & App Development service | 14 |
| Figure 5: Overview of the Graftor GUI..... | 16 |
| Figure 6: Extended DCAT model..... | 18 |
| Figure 7: Graftor GUI with an empty pipeline | 23 |
| Figure 8: Adding a new pipeline element | 23 |
| Figure 9: Defining a "drop-rows" pipeline element..... | 24 |
| Figure 10: Graftor "drop-rows" function element..... | 24 |
| Figure 11: Defining a "make-dataset" pipeline element..... | 24 |
| Figure 12: Defining a "derive-column" pipeline element | 25 |
| Figure 13: Defining custom functions | 25 |
| Figure 14: Defining a "mapc" pipeline element | 26 |
| Figure 15: Declaring namespace prefixes..... | 26 |
| Figure 16: Example Clojure output from a Graftor GUI pipeline | 27 |
| Figure 17: User authentication sequence..... | 28 |

List of Tables

| | |
|---|----|
| Table 1: Data Publisher (DP) requirements addressed | 9 |
| Table 2: Application Developer (AD) requirements addressed..... | 9 |
| Table 3: End User Data Consumer (EU) requirements addressed | 10 |
| Table 4: Application descriptions management | 18 |
| Table 5: Release descriptions management | 19 |
| Table 6: Simple dataset | 23 |

1 Introduction

This report represents supporting documentation for the prototype developed for Deliverable D2.2 “Open Data PaaS Prototype, v.1”. An online integrated version of the DaPaaS platform¹, including all available components from WP1-WP3 (D1.2, D2.2 and D3.2) is available to all consortium members. A publicly accessible version will be provided at the beginning of year 2, according to the plan.

The goal of this deliverable is to provide:

- The actual software prototype for an Open Platform-as-a-Service (PaaS) infrastructure, and integrate it with the DaaS prototype provided in WP1, thus providing an integrated DaPaaS platform.
- Detailed APIs specification and documentation for consumption of the core functionalities of the PaaS platform.

The PaaS prototype consists of a set of software components that were developed based on the requirements, design and architecture specification from Deliverable D2.1². The prototype aligns with Deliverable D1.2 in order to ensure seamless integration.

The rest of this report is structured as follows:

- Section 2 summarises how the prototype addresses the requirements from Deliverable D2.1.
- Section 3 describes the design and implementation of the software components that are integrated in the PaaS prototype.
- Section 4 describes deployment details for the DaPaaS platform.
- Section 5 outlines future work for the second version of the PaaS prototype.
- Appendix A provides a user guide for the Data Cleaning and Transformation graphical component developed as part of the PaaS prototype.
- Appendix B documents the APIs of the implemented PaaS software components.

¹ <http://dapaas.ontotext.com/demo/>

² <http://project.dapaas.eu/dapaas-reports/open-paas-requirements-design-architecture-specification>

2 Requirements Addressed by the Prototype

Deliverable D2.1 outlined a set of requirements for the different *roles* of the DaPaaS platform, i.e., Instance Operator, Data Publisher, Application Developer and End User Data Consumer.

In the first release of the DaPaaS platform, we have primarily focused on addressing the requirements for the Data Publisher as well as on requirements related to the Application Developer and the End User Data Consumer. The tables below summarize how the current version of the prototype as of M12 addresses the requirements for these three roles.

Table 1: Data Publisher (DP) requirements addressed

| ID | Name | Brief description of how the current version of the prototype addresses the requirement |
|-------|---|---|
| DP-01 | Dataset import | Addressed in D1.2. |
| DP-02 | Data storage & querying | Addressed in D1.2. |
| DP-03 | Dataset search & exploration | Addressed in D1.2. |
| DP-04 | Data interlinking | Addressed in D1.2. |
| DP-05 | Data cleaning & transformation | The Data Cleaning & Transformation service described in Section 3.3.1 provides the Data Publisher capabilities to apply simple data cleanup & transformation (incl. RDFization) over legacy data. |
| DP-06 | Dataset bookmarking & notifications | Addressed in D1.2. |
| DP-07 | Dataset metadata management, statistics & access policies | Addressed in D1.2. |
| DP-08 | Data scalability | Addressed in D1.2. |
| DP-09 | Data availability | Addressed in D1.2. |
| DP-10 | User registration & profile management | The User Management & Access Control service described in Section 3.2 allows users to register as a Data Publisher and gain access to the relevant DaaS services. |
| DP-11 | Secure access to platform | Addressed in D1.2. |
| DP-12 | UI for Data Publisher | Addressed in D3.2. |
| DP-13 | Data publishing methodology support | Addressed in D4.1. |

Table 2: Application Developer (AD) requirements addressed

| ID | Name | Brief description of how the current version of the prototype addresses the requirement |
|-------|---|---|
| AD-01 | Access to Data Publisher services (DP-01 – DP-13) | Addressed in D1.2. |
| AD-02 | Data export | Addressed in D1.2. |

| | | |
|-------|--|---|
| AD-03 | Develop applications in state-of-the-art programming languages | This first PaaS prototype allows Application Developers to develop Java-based web applications. For the second version of the prototype, the plan is to support additional programming languages. |
| AD-04 | Configure application deployment | Support for application deployment and configuration of common cloud resources, e.g. database/storage, is related to requirement AD-05 and will be further supported in the second version of the DaPaaS platform. Initial support for graphical widgets is supported in WP3 and will be further enhanced in the forthcoming Deliverable D3.3. |
| AD-05 | Deploy and monitor application | <p>The App Management & Deployment service described in Section 3.4 provides the Application Developer with an application hosting environment where data-intensive applications can be easily deployed. Tomcat is chosen as the initial application hosting environment for the DaPaaS platform. Thus, the current release supports Java-based web applications packaged as deployable war files.</p> <p>For the second version of the DaPaaS platform we are considering to use Docker³ technology in order to have support for other application environments and programming languages. Monitoring facilities for the deployed applications will be supported in the second version.</p> |
| AD-06 | Application metadata management, statistics & access policies | The App Catalog described in Section 3.5 allows Application Developers to describe metadata about their applications. Support for statistics and access policies will be added in the second version of the PaaS prototype. |
| AD-07 | UI for Application Developer | Addressed in D3.2. |
| AD-08 | Application development methodology support | Guidelines for Data Publisher is addressed in D4.1. These guidelines are a first step towards supporting the overall application development process. Further guidelines, specifically for the Application Developer, will be provided in the form of online user guides as part of the future work. |

Table 3: End User Data Consumer (EU) requirements addressed

| ID | Name | Brief description of how the current version of the prototype addresses the requirement |
|-------|---|--|
| EU-01 | User registration & profile management | The User Management & Access Control service described in Section 3.2 allows users to register as End User Data Consumers and manage their profiles. |
| EU-02 | Search & explore datasets and applications | Addressed in D1.2. |
| EU-03 | Datasets and applications bookmarking and notifications | Addressed in D1.2. |
| EU-04 | Mobile and desktop GUI access | Addressed in D3.2. |

³ <https://www.docker.com/whatisdocker/>

| | | |
|-------|--|--|
| EU-05 | Data export and download | Addressed in D1.2. |
| EU-06 | High availability of data and applications | <p>The DaPaaS platform provides a software-as-a-service infrastructure in which datasets and apps will be hosted by the DaPaaS project. This initial release of the DaPaaS platform caters primarily to Data Publishers and provides services that allow publishers to easily publish their datasets as RDF.</p> <p>To ensure high availability of data and applications we will follow a two-step strategy.</p> <ul style="list-style-type: none"> • The first step is to attract Data Publishers to register and use the platform so that high availability of data can be achieved. • The second step will be to attract Application Developers, who will create applications using the published data, ensuring high availability of applications. |

3 Design and Implementation of the Prototype

Figure 1 depicts the main software components, their relationships and associated APIs of the Platform Layer. The software components of the Platform Layer extend the capabilities offered by the Data Layer in five main service categories plus an administration service:

- **User Management & Access Control**, which manages user profiles and secure access control to apps and datasets.
- **Data Cleaning & App Development**, which provides functionalities for applications development, and support for data cleaning & transformation and data workflows.
- **Notification**, which provides functionality for subscribing to apps and datasets events and notifications.
- **App Management & Deployment**, which gives developers control over the deployed applications and configuration settings for the application-hosting environment.
- **Apps Catalog** for searching and exploring apps and their metadata.
- **Administration**, which allows the management and monitoring of the DaPaaS Platform, focusing on aspects related to the users, apps, datasets and services of the platform.

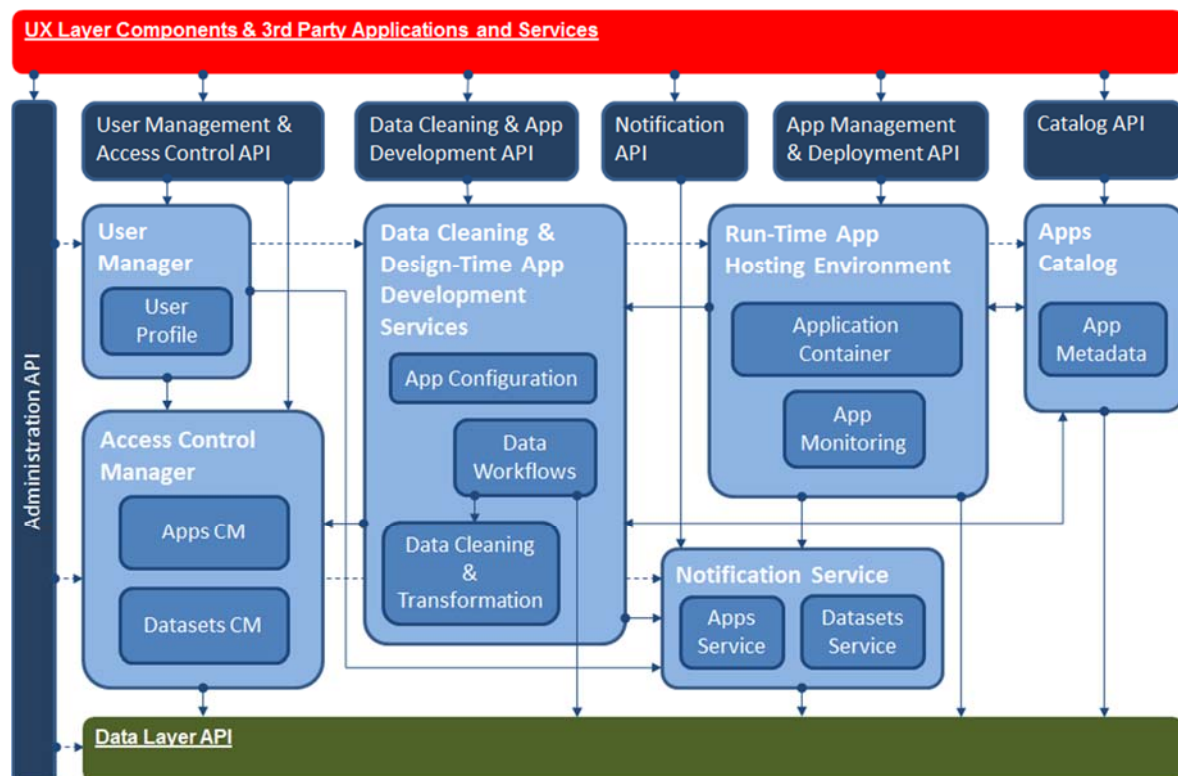


Figure 1: Architecture of the PaaS layer

3.1 Integration with the Data Layer

As can be seen from Figure 1 above, the components of the PaaS layer use the APIs of the DaaS layer (highlighted in Figure 2 below). The first version of the PaaS prototype makes use of APIs for:

- catalog, extended to support Applications;
- update, access and query for data access;
- import/export, specifically an adapter for the Grafter execution engine (which is also highlighted in Figure 2 below).

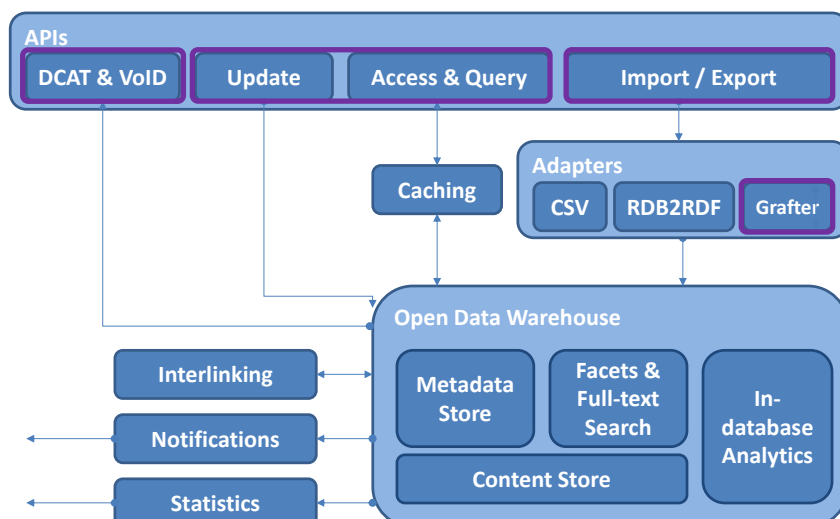
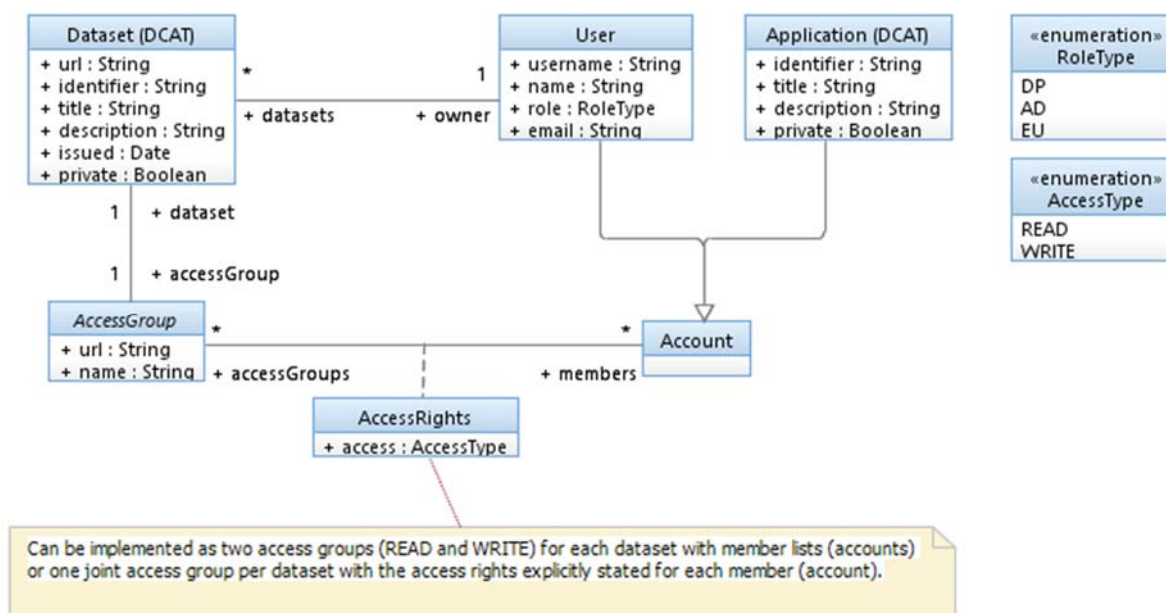


Figure 2: Architecture of the DaaS layer

3.2 User Management & Access Control

For the first version of the User Management & Access Control service we decided to simplify the access control as the primary role of the DaPaaS platform is to attract Data Publishers and Application Developers who are going to publish public datasets and host public apps ensuring that they are available to a wide range of users. Access control is needed to restrict the users that are allowed to update and modify the datasets.



3.3 Data Cleaning & App Development

The **Data Cleaning & App Development** service is a collection of data cleaning and design-time app development services:

- **App Configuration**, which is responsible for providing standardized mechanisms to configure cloud resources (e.g. data storage), DaaS services and UX components to be used by the deployed instance of the app at run-time.
- **Data Cleaning & Transformation**, which provides additional data management functionalities that complement Data Layer functionality with capabilities for data cleaning (duplicate removal), data transformation, as well as data mapping and alignment.
- **Data Workflows**, which provides the capability to define simple data-driven pipelines that use functionality from the Data Layer (e.g., import/export and publish data) and the added functionality offered by the Data Cleaning & Transformation component in order to support simple sequential data transformations.

Figure 4 illustrates the development scope for the first version of the Data Cleaning & App Development service. After an analysis of existing Data Cleaning & Transformation solutions, Grafter⁴ (developed in WP4) was chosen as the underlying framework, and will be extended with RESTful services to support the Data Workflows component (developed in WP2).

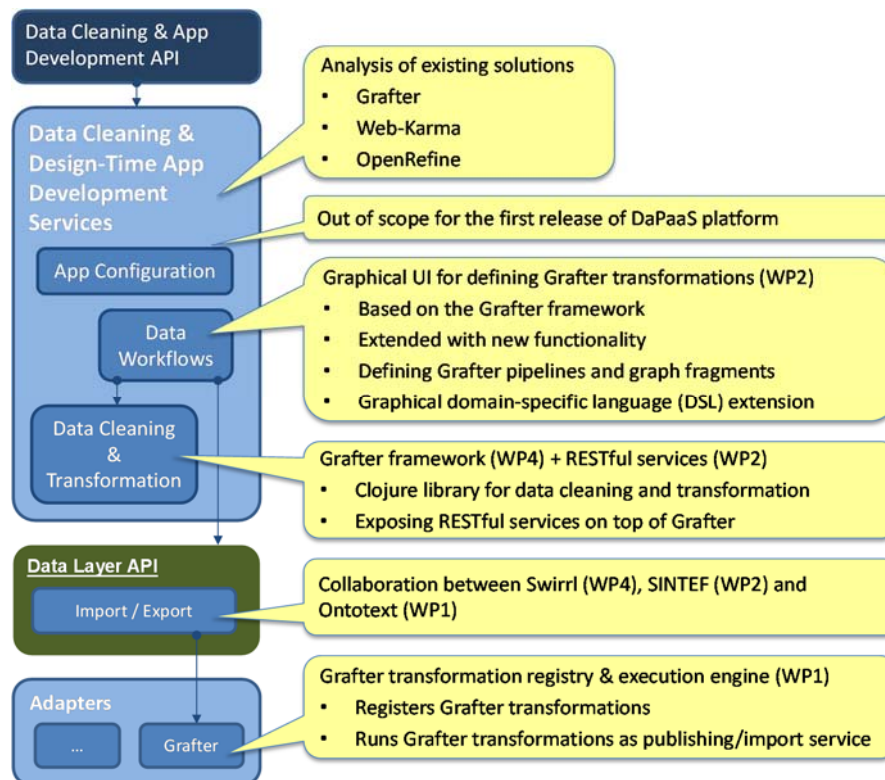


Figure 4: Development scope for first version of the Data Cleaning & App Development service

The Data Workflows component provides a graphical UI for defining a *Graftor transformation*, which is a simple *data workflow* consisting of two main steps⁵:

1. **Specify a pipeline**, of tabular transformations for data cleaning and transformation.
2. **Create the graph fragments**, resulting in the generation of an RDF graph.

The resulting *Graftor transformation* is registered as a publishing/import service on the DaPaaS platform (WP1) for the Data Publishers to use.

⁴ <https://github.com/swirrl/grafter>

⁵ <http://grafter.org/example/index.html>

3.3.1 Data Cleaning & Transformation

In Deliverable D2.1 we provided an initial analysis of data integration solutions, where Web-Karma⁶ and OpenRefine⁷ were identified as two promising open source solutions, on which to base the development of the Data Cleaning & Transformation service in the PaaS prototype. Following this initial analysis it became apparent that the Grafter framework planned in WP4 of DaPaaS could also be a promising candidate. It provided a good opportunity for a closer collaboration between partners in the project.

When a first release of the Grafter prototype became available, we did a further analysis of the solutions, comparing Grafter to amongst others Web-Karma and OpenRefine. The analysis compared the different data input, cleaning and transformation features of the solutions, i.e.:

- Dataset operations
- Column operations
- Row operations
- RDF related features

The spreadsheet in Appendix C documents the comparison of the features. Note that the comparison distinguishes between Grafter, Incanter and Grafter+Incanter. Incanter⁸ is a modular Clojure library with many features (e.g., mathematical functions, statistical functions, data manipulation functions, etc.) of which the data manipulation functions are used by the Grafter framework.

While Web-Karma and OpenRefine are two promising solutions, our analysis shows that there is a tight coupling of UI and backend functionality that hinders distributions, and furthermore makes it difficult to refactor the source code to expose programmatic service APIs. Grafter was designed to provide a separate backend library of functions, on top of which there could be an independent graphical UI.

3.3.2 Data Workflows

Since Grafter does not currently provide a graphical user interface, we decided to build upon it and thus implement the Data Workflow component as. This yields several advantages:

- Grafter already covers 90+ % of the functionality in Web-Karma and OpenRefine.
- Can easily define a graphical DSL on top of the available APIs.
- Able to expose programmatic and service APIs for batch processing and enable distribution.
- Able to expose more functions as GUI elements. (Suggested icons for some functions are shown in the spreadsheet in Appendix C.)

The Grafter GUI is a graphical wrapper over the Grafter library that is used to define a *Grafter transformation data workflow* and its implementation as Clojure code. This transformation can then be executed over any number of input datasets that conform to the same schema, in order to eventually transform them from their tabular representation (in CSV or Excel spreadsheet format) into semantic data (in RDF).

⁶ <http://www.isi.edu/integration/karma/>

⁷ <http://openrefine.org/>

⁸ <http://incanter.org/>

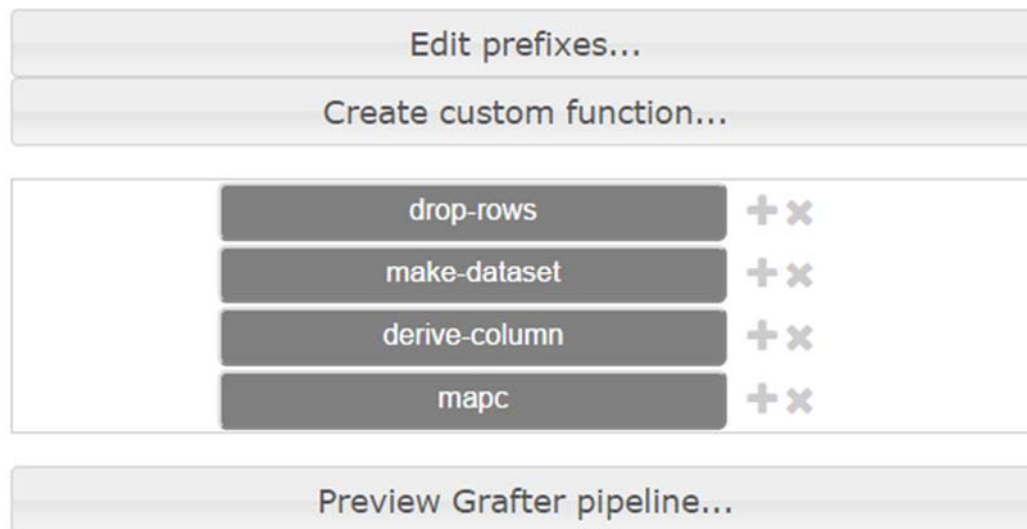


Figure 5: Overview of the Grafter GUI

Currently, the Grafter GUI supports the creation of a *Grafter transformation* through the following features:

- Specify the *Grafter pipeline*
 - user-defined Clojure functions – can be used in the pipeline to transform the data
 - pre-defined Grafter APIs – a set of APIs that have been wrapped into UI elements.
- Create the *graph fragments*
 - defining namespace prefixes – used as shorthand when building the RDF representation

For more details, please see the example transformation in Appendix A.

The development of the Grafter GUI is made available as open source software under the Eclipse Public License v1.0 at GitHub⁹. The implementation of the Grafter GUI uses the following technologies and frameworks:

- **jQuery**¹⁰ and **jQuery UI**¹¹ are used for user interface elements (dialogs, buttons, selectors, etc.)
- **appendGrid**¹² is used for implementing the table-based input.
- **Font Awesome**¹³ is used to depict some of the symbols in the graphics.
- **CodeMirror**¹⁴ is used for code editing, and syntax highlighting of the Clojure code. Eventually, we are also planning to implement autocompletion.
- **jsedn**¹⁵ is used for generation of the Grafter/Clojure code, which defines the Grafter transformation.
- **Grafter** is used for running the output code from the GUI.
- **Leiningen**¹⁶ is used for building the Clojure project and creating the corresponding JAR file.

⁹ <https://github.com/dapaas/grafter-gui>

¹⁰ <http://jquery.com/>

¹¹ <http://jqueryui.com/>

¹² <https://appendgrid.apphb.com/>

¹³ <http://fortawesome.github.io/Font-Awesome/>

¹⁴ <http://codemirror.net/>

¹⁵ <https://github.com/shaunxcode/jsedn>

3.4 Applications Management & Deployment

This first PaaS prototype allows Application Developers to develop Java-based web applications and deploy them on the platform infrastructure. For the second version of the prototype, we will support additional programming languages.

The App Management & Deployment service provides the Application Developer with an application hosting environment where data-intensive applications can be easily deployed. Apache Tomcat is chosen as the initial application hosting environment for the DaPaaS platform. Thus, the current release supports Java-based web applications packaged as deployable war files.

Tomcat offers convenient support for remote management of all phases of the applications lifecycle: deployment, starting, stopping, undeployment, status checking, etc. without a need for restarting the environment. On top of this management layer, we built services abstracting the actual deployment details, tightly integrated with the *Application Catalog* services. Thus some of the routines are happening in the background while the *Application Developer* is managing the metadata descriptions.

In fact, the only direct interaction of the user with the deployment services is when a new web application (.war file) has to be deployed. This can be achieved by using the *Application deployment API* (Section 7.3) where the implementation takes care of:

- the deployment of the application,
- starting the application, and
- registering it in the apps catalogs.

The removal of metadata descriptions from the applications catalog is integrated within the undeployment functionality of Tomcat, so the user does not have to perform any additional cleaning up.

For the second version of the DaPaaS platform we are considering using Docker technology in order to provide support for other application environments and programming languages. Monitoring facilities for the deployed applications is also planned to be supported in the second version.

3.5 Applications Catalog

On the metadata level the datasets and applications share many common properties, which is the reason we adopted the DCAT vocabulary here as well. To cover the specific applications aspects we added additional extensions to the model and the metadata vocabulary. D1.2 provides an overview of the W3C's DCAT recommendation and its direct applicability to datasets. Here we focus on the extended model for application descriptions. Figure 6 highlights the extensions proposed and implemented in the first prototype of the platform.

¹⁶ <http://leiningen.org/>

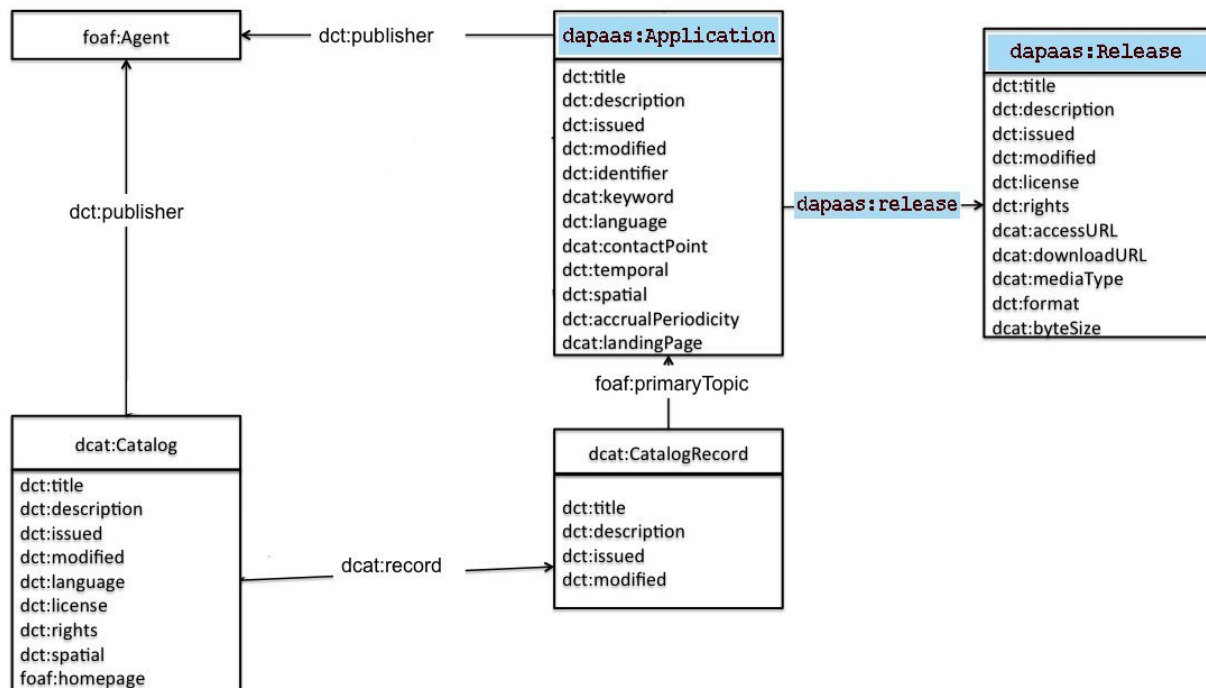


Figure 6: Extended DCAT model

Taking the same approach as with dataset having different distributions, here we introduce two new classes:

- **dapaas:Application** - a general description of an application, abstracting the concrete versions, access mechanisms, etc.
- **dapaas:Release** - a description of a specific version of the application in terms its concrete realisation (access and download URLs, versioning specification, etc.).

Each application may refer to one or more releases via the newly introduced property **dapaas:release**. It connects the abstract descriptions with the consumption point of applications' functionalities.

The extended parts reuse as many properties as possible from their counterparts in the original DCAT model.

Table 4 and Table 5 provides an overview of the API functionality. Detailed API documentation and the underlying implementations are described in Section 7.4. The functionalities in terms of API design are similar to the ones from the datasets catalog (D1.2).

Table 4: Application descriptions management

| Method | URL | Description |
|--------|--------------------------------------|---|
| GET | /users/{user-id}/appcatalog | List a catalog of applications for the user (owned and shared) |
| GET | /users/{user-id}/applications | Get details about certain application (meta data) |
| POST | /users/{user-id}/applications | Create a new application description |
| PUT | /users/{user-id}/applications | Update an existing application description |
| DELETE | /users/{user-id}/applications | Delete an application (including all releases) |
| GET | /users/{user-id}/applications/search | Search for applications based on meta-data (titles, descriptions, keywords) |

Table 5: Release descriptions management

| Method | URL | Description |
|--------|---------------------------|------------------------------------|
| GET | /users/{user-id}/releases | Get full description of a release |
| POST | /users/{user-id}/releases | Create a new release (description) |
| PUT | /users/{user-id}/releases | Update an existing release |
| DELETE | /users/{user-id}/releases | Delete a release |

4 Deployment Details

The first version of the WP2 PaaS prototype is hosted together with the WP1 DaaS prototype, thus forming an integrated DaPaaS platform. The DaPaaS platform runs on two machines as described in Deliverable D1.2, one high-performance machine designated for the data warehouse with 64GB RAM, and a second moderate-performance machine for the DaaS and PaaS services running the operating system Ubuntu 12.04.2 LTS. Additionally for application deployment & execution environment we use a separate machine with moderate characteristics (16GB RAM, Intel i7 CPU). Its only purpose is to run third party web applications (*Application Developers* support) in an isolated environment.

The following 3rd party software components are required to support the PaaS layer of the DaPaaS platform:

- Apache Tomcat 7.0 (applications container)
- Apache CXF framework (web services development framework)

The PaaS layer of the platform consists of several components that are packaged as web applications or exposed as RESTful web services:

- User management and access control
- Data cleaning & transformation service which consists of three main components
 - Grafter GUI client component (JavaScript and HTML5)
 - Grafter server component (creates jar file for Grafter pipeline and registers it)
 - Grafter execution engine (described in D1.2)
- Catalog services

5 Future Work

For the second release of the DaPaaS platform we will add further functionality addressing the needs of the Application Developer and the End User Data Consumer, while at the same time strengthening the capabilities offered to the Data Publisher.

5.1 Docker (extending the App Hosting Environment)

The Applications Management & Deployment service described in Section 3.4 provides the Application Developer with an application hosting environment where data-intensive applications can be easily deployed. Tomcat is chosen as the initial application hosting environment for the DaPaaS platform. For the second version of the DaPaaS platform we are considering to use Docker.

Docker¹⁷ defines a format for bundling an application and all its dependencies into a single object which can be transferred to any Docker-enabled machine, and executed there with the guarantee that the execution environment will be the same. This allows the packaged applications to run in different environments without being reconfigured again. In addition the Docker tool offers features for application deployment, automatic build, versioning and component re-use. Docker also provides an API¹⁸ for automating and customizing the creation and deployment of containers.

5.2 Notifications

Support for notifications were not part of the first release since it was regarded as “nice to have” but not critical. This decision was taken in order to focus on more important service capabilities aimed at the Data Publisher. For the second release of the DaPaaS platform we will implement a notification service that supports:

- Publishing and subscribing different types of events and notifications for datasets.
- Publishing and subscribing events and notifications for users and apps.

5.3 Grafter GUI enhancements (Data Cleaning & Workflows)

The first version of the Grafter GUI and the corresponding Grafter services provide a Data Publisher with functionality to define and deploy a Grafter pipeline for publishing datasets based on tabular data (i.e., CSV files and Excel spreadsheets) as RDF data in the DaPaaS platform. Improvements to the Grafter GUI include two important graphical features:

- **Preview functionality:** In order to provide better support for defining a proper Grafter pipeline, it is useful to see the effects of each step, i.e., the result of each Grafter function as they are applied. This will be implemented as a preview window, showing the changes to the tabular data (e.g., displaying the new column with the transformed data). By stepping (forward/backward) through the pipeline functions the effect of the selected function in the pipeline are shown directly to the user. This makes it easier for the user to verify that the pipeline being defined is correct.
- **Mapping to RDF:** After the pipeline is defined you use the Grafter graph function to translate the data table created by the pipeline into a sequence of RDF statements. A graphical UI for defining the RDF graph fragments will be implemented. This UI may be extended to support the loading of one or more ontologies that are used in the generation of the RDF graph.

In addition to the two graphical features described above, we also plan to add support for:

- **Graphical domain-specific language (DSL):** The DSL will allow users to express multiple Grafter pipelines (with common “sublines”) as a graphical model. In order to ensure that the graphical DSL will be usable, we will focus on simplicity, e.g., no support for conditional branching or other complex data workflow operations. The idea is to keep it simple, with input/output that can be fed into different pipelines.

¹⁷ <https://www.docker.io/>

¹⁸ <http://docs.docker.io/en/latest/api/>

5.4 Management UI

For the first prototype the setup and configurations are done manually, by installing and running different services and inspecting various system logs for system status. The system health and performance monitoring is also partially concerned. For the next version of the platform, the *Instance operator* will be supported by a unified UI management console providing single place for exploring different aspects of the platform as well as ability to perform different management routines - system checks and monitoring, usage quota enforcements, resources provisioning, etc.

6 Appendix A: User Guide for Grafter GUI

The Grafter GUI can be used to define data pipelines based on the Grafter library, which are used to perform data transformation.

As an example we will use a simple dataset that consists of three columns - *name*, *sex* and *age*:

Table 6: Simple dataset

| Name | Sex | Age |
|-------|-----|-----|
| Alice | f | 34 |
| Bob | m | 63 |

The Grafter GUI main elements are shown on the following figure:

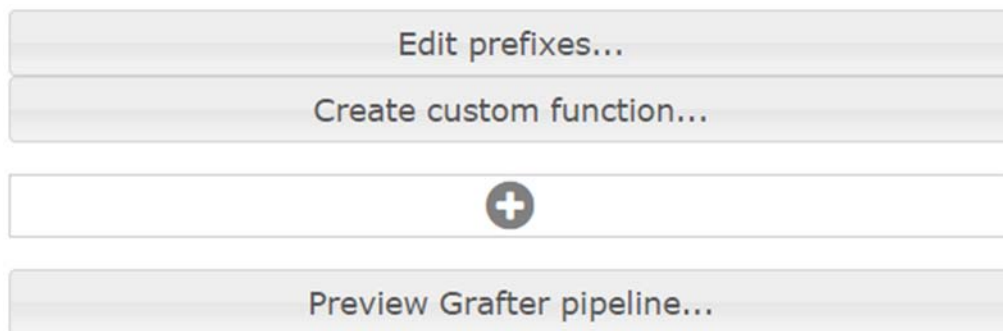


Figure 7: Grafter GUI with an empty pipeline

The data transformation pipeline consists of several elements that are defined using the '+' button in the interface. Clicking it prompts the following dialog:

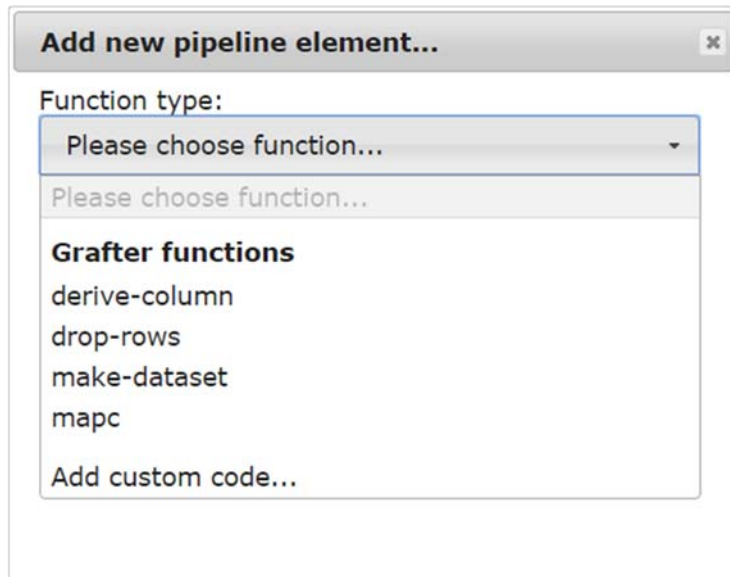


Figure 8: Adding a new pipeline element

The dialog provides users a choice of several pre-defined Grafter functions that are supported by UI elements. They can be used to easily and quickly define steps from the pipeline. For example, the 'drop-rows' Grafter function is shown in the following:

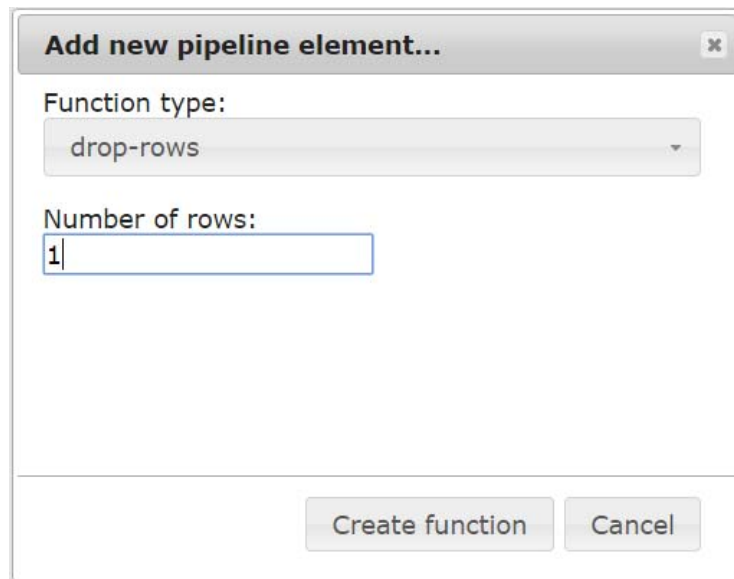


Figure 9: Defining a "drop-rows" pipeline element

The function can be used to delete a certain number of rows from the input dataset. In this case, users need to only specify that number.

In our case, we need to delete the first row that contains the column names as it will not be used later on when we create the RDF representation of the data.

Selecting the 'Create function' button will create a new element that corresponds to the selected Graftor function or custom code. In the case of the drop-rows, it is as shown in the following image:

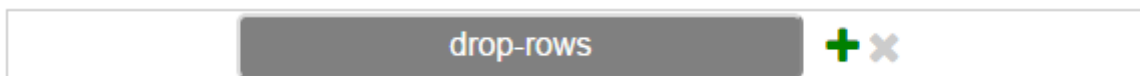


Figure 10: Graftor "drop-rows" function element

Other supported Graftor APIs are:

- "make-dataset" – maps a set of columns of a dataset to a set of column names. The column names are surrounded by square brackets ('[', ']') where each output column name begins with a colon (':').

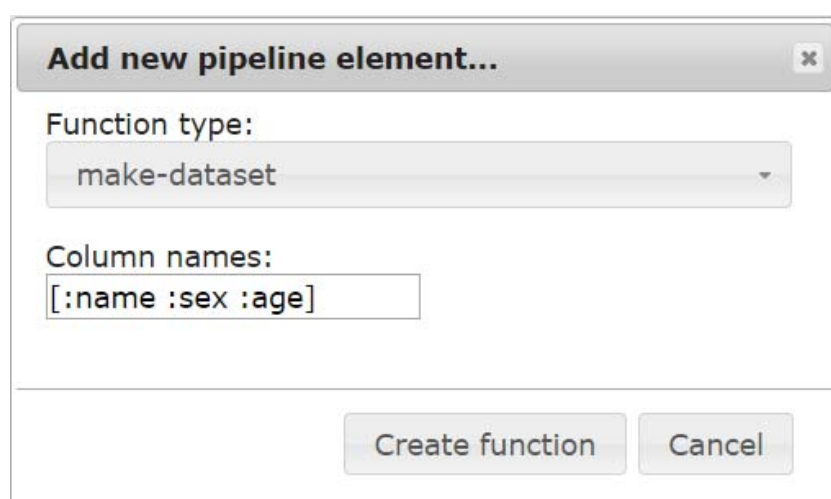


Figure 11: Defining a "make-dataset" pipeline element

In our example, we will use the make-dataset to set up aliases (":name", ":sex", and "age") of all of the columns that we use in the transformation (which are in fact all three of them).

- "derive-column" – creates a new output column which is derived from a set of input columns using a specified function (defined by the user) or using a prefix (when 'URI-ifying' literals).

Add new pipeline element...

Function type:
derive-column

Name of derived column:
:person-uri

Columns to derive from:
[:name]

Select function:
base-id

Create function Cancel

Figure 12: Defining a "derive-column" pipeline element

For the purposes of our example, we will create a new column, denoted by ":person-uri" where we URI-ify the input people's names using a prefix "base-id". Prefixes are defined using the "Edit prefixes..." dialog, an example of which is shown later on in the tutorial.

In case we need any other custom functions, we can define them through the use of another dialog, where users can input Clojure code:

Define custom function...

Create new function...

Code:

```
1 (defn ->gender
2   [str]
3   {
4     "f" (s "female")
5     "m" (s "male")
6   })
7 }
```

Save Done

Define custom function...

->integer

Code:

```
1 (defn ->integer
2   "An example transformation function that converts a
3   string to an integer"
4   [s]
5   (Integer/parseInt s))
```

Save Done

Figure 13: Defining custom functions

This dialog can be invoked from selection boxes when adding new pipeline elements (available for each element where this is needed), or from the "Create custom function..." button. In our case, we will define two functions – "->gender", and "->integer". The former is used to map the letter denoting the gender of the person, to its full name. The latter is used to parse a string into an integer.

- "mapc" – for each row in the dataset, applies a function to a particular column (defined by a key). Each column name is preceded by a colon (':').

Add new pipeline element...

Function type:
mapc

Map columns to functions

| Column key | Function |
|------------|-----------|
| :age | Choose... |
| :sex | Choose... |

Existing functions
base-id
->integer
->gender
Create custom function...

Create function Cancel

Figure 14: Defining a "mapc" pipeline element

The above picture shows how we associate the custom functions we already defined with the according column keys denoting the columns containing the ages of people (":age"), and their gender (":sex").

Through the "Edit prefixes..." dialog, users are able to define the prefixes that are eventually used as shorthand when building the RDF graph or that are used in the pipeline to URI-ify our data. They are defined in a table, where a prefix name is associated with an according URI, as shown in the following figure:

Define RDF prefixes

| Prefix name | URI |
|-------------|-----------------------------|
| base-domain | http://my-domain.com |
| base-graph | http://my-domain.com/graph/ |
| base-id | http://my-domain.com/id/ |
| base-vocab | http://my-domain.com/def/ |
| base-data | http://my-domain.com/data/ |

Append Row

Save Cancel

Figure 15: Declaring namespace prefixes

Each pipeline definition corresponds to a certain Clojure representation (which also includes the Grafter library and other miscellaneous imports). The output Clojure code can be previewed by clicking the "Preview Grafter pipeline..." button:

Preview Grafter pipeline...

```

1 datasets make-dataset move-first-row-to-header _]] [grafter.rdf.sesame
2 :as ses] [grafter.rdf.ontologies.rdf :refer :all]
3 [grafter.rdf.ontologies.foaf :refer :all] [grafter.rdf.ontologies.void
4 :refer :all] [grafter.rdf.ontologies.dctterms :refer :all]
5 [grafter.rdf.ontologies.vcard :refer :all] [grafter.rdf.ontologies.pmd
6 :refer :all] [grafter.rdf.ontologies.qb :refer :all]
7 [grafter.rdf.ontologies.os :refer :all] [grafter.rdf.ontologies.sdmx-
8 measure :refer :all]))
9
10 (def base-domain (prefixer "http://my-domain.com"))
11 (def base-graph (prefixer "http://my-domain.com/graph/"))
12 (def base-id (prefixer "http://my-domain.com/id/"))
13 (def base-vocab (prefixer "http://my-domain.com/def/"))
14 (def base-data (prefixer "http://my-domain.com/data/"))
15
16 (defn base-id (prefixer (base-domain "/id/")))
17 (defn ->integer "An example transformation function that converts a
18 string to an integer" [s] (Integer/parseInt s))
19 (defn ->gender [str] {"f" (s "female") "m" (s "male")})
20
21 (defn pipeline [dataset] (-> dataset (drop-rows 1) (make-dataset [:name
22 :sex :age]) (derive-column :person-uri [:name] base-id) (mapc {"age" -
23 >integer ":sex" ->gender})))

```

Figure 16: Example Clojure output from a Grafter GUI pipeline

7 Appendix B: API Documentation

7.1 User Management API

7.1.1 Results

All services return some meaningful HTTP status code along with the result - mostly 40x for erroneous requests, 20x for successful execution

7.1.2 User Services & Web UI Authentication

Authentication is performed via HTTP form-based authentication. The authentication flow goes like this:

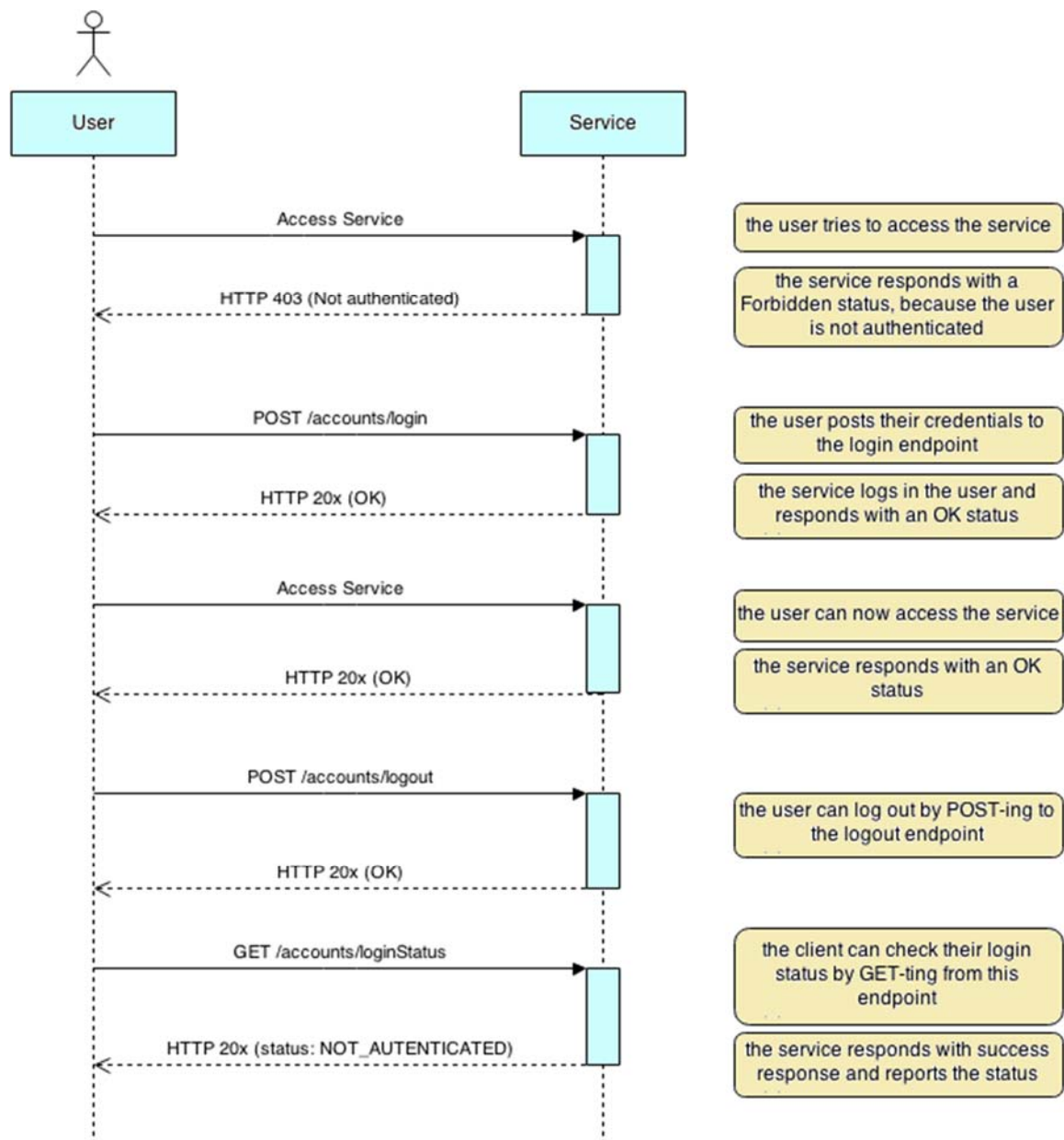


Figure 17: User authentication sequence

7.1.3 Account Service

7.1.3.1 Login with username and password

```
PUT /accounts/login

BODY:
{
  "username" : "<username>",
  "password" : "<password>",
}

RESULT
OK //SUCCESS
{"message" : "<error_message>"} //ERROR
```

7.1.3.2 Login with Google+ ID

```
PUT /accounts/login

BODY:
{
  "google_id" : "<google_id>",
}

RESULT
OK //SUCCESS
{"message" : "<error_message>"} //ERROR
```

7.1.3.3 Login with Twitter ID

```
PUT /accounts/login

BODY:
{
  "twitter_id" : "<twitter_id>",
}

RESULT
OK //SUCCESS
{"message" : "<error_message>"} //ERROR
```

7.1.3.4 Login with Facebook ID

```
PUT /accounts/login

BODY:
{
  "facebook_id" : "<facebook_id>",
}

RESULT
OK //SUCCESS
{"message" : "<error_message>"} //ERROR
```

7.1.3.5 Signup (create new account)

```
POST /accounts/

BODY:
{
```

```
"username" : "<login name>",  
"password" : "<pass>",  
"role" : "<dapaas_role>",  
"name" : "<User name>",  
"email" : "<email>"  
}  
  
RESULT  
OK //SUCCESS  
{ "message" : "<error_message>" } //ERROR
```

7.1.3.6 Signup (create new account) with Facebook

```
POST /accounts/  
  
BODY:  
{  
  "facebook_id" : "<facebook_id>",  
  "role" : "<dapaas_role>",  
  "name" : "<User name>",  
  "email" : "<email>"  
}  
  
RESULT  
OK //SUCCESS  
{ "message" : "<error_message>" } //ERROR
```

7.1.3.7 Signup (create new account) with Twitter

```
POST /accounts/  
  
BODY:  
{  
  "twitter_id" : "<twitter_id>",  
  "role" : "<dapaas_role>",  
  "name" : "<User name>",  
  "email" : "<email>"  
}  
  
RESULT  
OK //SUCCESS  
{ "message" : "<error_message>" } //ERROR
```

7.1.3.8 Signup (create new account) with Google+

```
POST /accounts/  
  
BODY:  
{  
  "google_id" : "<google_id>",  
  "role" : "<dapaas_role>",  
  "name" : "<User name>",  
  "email" : "<email>"  
}  
  
RESULT  
OK //SUCCESS  
{ "message" : "<error_message>" } //ERROR
```

7.1.3.9 Logout

```
PUT /accounts/logout

RESULT
OK //SUCCESS
{"message" : "<error_message>"} //ERROR
```

7.1.3.10 Login Status

```
GET /accounts/login_status

RESULT
{"status": "AUTHENTICATED"}
or
{"status": "NOT_AUTHENTICATED"}
```

7.1.3.11 Get details

```
GET /accounts/details

RESULT:
{
  "username" : "<login name>",
  "email" : "<email>",
  "name" : "<user name>",
  "role" : "<dapaas_role>",
  "phone_num" : "<phone_num>",
  "addr_1" : "<addr_1>",
  "addr_2" : "<addr_2>"
}
```

7.1.3.12 Add/Update details

```
PUT/PATCH /accounts/details

BODY:
{
  "username" : "<login name>",
  "email" : "<email>",
  "name" : "<user name>",
  "role" : "<dapaas_role>",
  "phone_num" : "<phone_num>",
  "addr_1" : "<addr_1>",
  "addr_2" : "<addr_2>"
}

RESULT
OK //SUCCESS
{"message" : "<error_message>"} //ERROR
```

7.1.3.13 Change password

```
PUT /accounts/password

BODY:
{
  "new_password" : "<password>"
}

RESULT
```

```
OK //SUCCESS
{"error_message" : "<message>"} //ERROR
```

7.1.3.14 Password reset request

```
POST /accounts/password/reset

BODY:
{
  "email" : "<email>"
}

RESULT
OK //SUCCESS
{"error_message" : "<message>"} //ERROR
```

7.1.3.15 Password reset confirmation

```
PUT /accounts/password/confirm

BODY:
{
  "email" : "<email>",
  "new_password" : "<password>",
  "token" : "verification token"
}

RESULT
OK //SUCCESS
{"error_message" : "<message>"} //ERROR
```

7.1.4 API Key Management

7.1.4.1 Get API Keys

```
GET /api_keys/

BODY:
[
  {"api_key" : "<api_key>", "enabled" : true},
  ...
]
```

7.1.4.2 New API Key

```
POST /api_keys/

RESULT
{"api_key" : "<api_key_id>", "secret" : "<api_key_secret>"} //SUCCESS
{"error_message" : "<message>"} //ERROR
```

7.1.4.3 Enable API Key

```
PUT /api_keys/<api_key>/enable

RESULT
OK //SUCCESS
{"error_message" : "<message>"} //ERROR
```


7.1.4.4 Disable API Key

```
PUT /api_keys/<api_key>/disable

RESULT
OK //SUCCESS
{"error_message" : "<message>"} //ERROR
```

7.1.4.5 Delete API Key

```
DELETE /api_keys/<api_key>

RESULT
OK //SUCCESS
{"error_message" : "<message>"} //ERROR
```

7.2 Data Cleaning & Transformation API

This section is identical to the one in the annexes of D1.2 as this API is related to the two work packages.

7.2.1 Transformation application API (Grafter Import)

7.2.1.1 Apply transformation

| | |
|--------------------------|---|
| URL | /dapaas-import-services/grafter/apply |
| HTTP Method | POST |
| Description | Performs CSV-2-RDF data transformation with Grafter and loads the result in certain repository |
| Inputs (HTTP headers) | publisher - the publisher ID from the catalogs repository-url - target repository to store the result repository-graph - optional graph in the repository transformation-id - the id of the specific Grafter transformation to be used CSV data upload as multipart attachment file - the CSV content name - the original file name of the CSV file |
| Response | Operation completion status (HTTP response code) |

7.2.2 Transformations management API

7.2.2.1 Register transformation

| | |
|-------------|---|
| URL | /dapaas-import-services/grafter/register |
| HTTP Method | POST |
| Description | Registers a new transformation into the DaPaaS platform |
| Inputs | HTTP headers: transformation-id - the id for the new Grafter transformation description - free text description of the transformation, what it does, what it requires, etc. jar binary as multipart attachment file - the jar content name - the original file name of the jar file (it is not necessary to be the same) |

| | |
|----------|--|
| | as the ID) |
| Response | Operation completion status (HTTP response code) |

7.2.2.2 List transformations

| | |
|-------------|--|
| URL | /dapaas-import-services/grafter/list |
| HTTP Method | GET |
| Description | Lists all available (registered) transformations in the platform |
| Inputs | - |
| Response | JSON array of objects representing registered transformations (name and description). Example: [{ "id" : "myGrafter1", "descr": "This is my transformation text" }, ...] |

7.2.2.3 Unregister transformation

| | |
|-------------|---|
| URL | /dapaas-import-services/grafter/unregister |
| HTTP Method | DELETE |
| Description | Unregisters a transformation from the platform |
| Inputs | HTTP header: transformation-id - the id of the specific Graftor transformation to be unregistered (removed) |
| Response | Operation completion status (HTTP response code) |

7.3 Applications Deployment API

7.3.1 Deploy application API

7.3.1.1 Deploy application

| | |
|--------------------------|--|
| URL | /appdeploy |
| HTTP Method | POST |
| Description | Deploys a web application packaged as a WAR file, belonging to certain release of an application from the catalog. |
| Inputs (HTTP headers) | release - the release URI from the metadata new-url-hint - preferred new URL endpoint suffix. The implementation checks for collisions and generates a new unique name derived from this parameter value. a) remote war for deployment: app-url - location of the war file on the Web + credentials (if it is necessary) |

| | |
|----------|--|
| | app-user - user name app-pass - password b) war upload as multipart attachment (local file) file - the content name - the original file name of the war file |
| Response | The new application deployment URL like: {"@id":"http://project.dapaas.eu/applications/newapp"} Note: the service implementation is responsible to update the release meta data property: <i>accessURL</i> |

7.4 Applications Catalog API

7.4.1 Catalog access API

7.4.1.1 Get application catalog

| | |
|-------------|---|
| URL | /users/{user-id}/appcatalog |
| HTTP Method | GET |
| Description | List user's applications as well as public or shared ones with the user |
| Inputs | user-id - the owner id HTTP headers: Accept - serialization format for the result: <ul style="list-style-type: none"> • application/ld+json • application/rdf+xml or any RDF type showShared - includes public and shared applications in the result as well. Valid values are 'y' and 'n' |
| Response | List of application catalog records using the extended DCAT vocabulary in RDF or JSON-LD |

7.4.1.2 Search applications

| | |
|-------------|---|
| URL | /users/{user-id}/applications/search |
| HTTP Method | GET |
| Description | Search for applications by keywords on meta data (titles, descriptions, etc.) |
| Inputs | user-id - the owner id HTTP header: Accept - serialization format for the result: <ul style="list-style-type: none"> • application/ld+json • application/rdf+xml or any RDF type URL parameter: q - the search query expression (plain text) |
| Response | List of application catalog records using the extended DCAT vocabulary in RDF or JSON-LD |

7.4.2 Applications descriptions access and management API

7.4.2.1 Get application description

| | |
|-------------|--|
| URL | /users/{user-id}/applications |
| HTTP Method | GET |
| Description | Get a application description (meta data) by id |
| Inputs | user-id - the owner id URL parameter: id - URI of the application, taken from the catalog HTTP header: Accept - result serialization format |
| Response | Complete application description using the extended DCAT vocabulary in RDF or JSON-LD |

7.4.2.2 Create new application description

| | |
|-------------|--|
| URL | /users/{user-id}/applications |
| HTTP Method | POST |
| Description | Create a new application description |
| Inputs | HTTP headers: user-id - the owner id Content-Type - format of the data supplied Application description as RDF or JSON-LD as request body Note: if the description contains no @id, the system will generate one |
| Response | URI of the new application in the format: { "@id" : "http://dapaas.eu/application/app1" } |

7.4.2.3 Update application description

| | |
|-------------|---|
| URL | /users/{user-id}/applications |
| HTTP Method | PUT |
| Description | Update an existing application description |
| Inputs | user-id - the owner id HTTP header: Content-Type - format of the data supplied Application description as RDF or JSON-LD as request body |
| Response | Operation completion status (HTTP response code) |

7.4.2.4 Delete application description

| | |
|-------------|--|
| URL | /users/{user-id}/applications |
| HTTP Method | DELETE |
| Description | Delete an application description with all of its releases |
| Inputs | user-id - the owner id |

| | |
|----------|---|
| | HTTP header: id - URI of the application to be removed |
| Response | Operation completion status (HTTP response code) |

7.4.3 Releases management API

7.4.3.1 Get release description

| | |
|-------------|--|
| URL | /users/{user-id}/releases |
| HTTP Method | GET |
| Description | Get a release description by id (contained in the corresponding application description) |
| Inputs | user-id - the owner id URL parameter: id - URI of the release, taken from the application description HTTP header: Accept - result serialization format |
| Response | Complete release description using the extended DCAT vocabulary in RDF or JSON-LD |

7.4.3.2 Create new release description

| | |
|-------------|---|
| URL | /users/{user-id}/releases |
| HTTP Method | POST |
| Description | Create a new release description |
| Inputs | user-id - the owner id URL parameter: app - URI of the application containing the release HTTP header: Content-Type - data input format release description as RDF or JSON-LD in the request body Note: if the description contains no @id, the system will generate one |
| Response | URI of the new release in the format: { "@id" : "http://dapaas.eu/release/v123." } |

7.4.3.3 Update release description

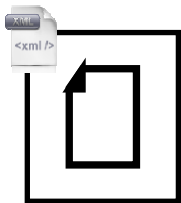
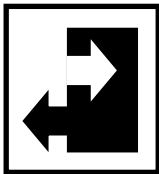
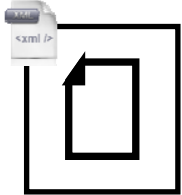
| | |
|-------------|--|
| URL | /users/{user-id}/releases |
| HTTP Method | PUT |
| Description | Update an existing release description |
| Inputs | user-id - the owner id URL parameter: app - URI of the dataset containing the distribution HTTP header: |

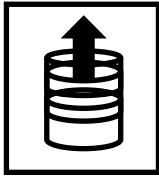
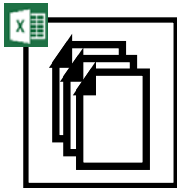
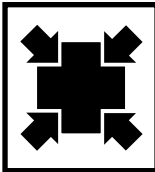
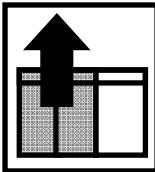
| | |
|----------|--|
| | Content-Type - data input format Release description as RDF or JSON-LD in the request body |
| Response | Operation completion status (HTTP response code) |

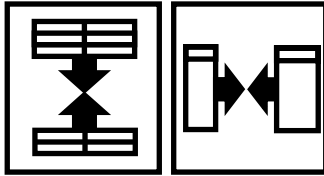
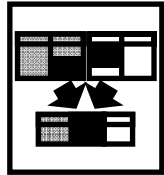
7.4.3.4 Delete release description

| | |
|-------------|--|
| URL | /users/{user-id}/releases |
| HTTP Method | DELETE |
| Description | Delete a release description |
| Inputs | user-id - the owner id URL parameter: distrib - URI of the release to be deleted |
| Response | Operation completion status (HTTP response code) |

8 Appendix C: Comparison of Data Cleaning & Transformation solutions

| | | Feature | Grafter | Incanter | Icons | Grafter+Incanter | Web-Karma | OpenRefine/GRefine |
|---|----------|---|--|----------|---|--|------------------------|--|
| Data input, cleaning and transformation | datasets | load a dataset from CSV or XLSX | open-all-datasets | |  | open-all-datasets | Import -> From File | Create Project (This Computer) -> Choose Files |
| | | load dataset from XML, JSON | | | | | Import -> From File | Create Project (This Computer) -> Choose Files |
| | | load dataset from RDF/XML, RDF/N3, ODF, text (fixed width, line-based), PC-Axis | statements (supports all RDF serializations) | | | statements (supports all RDF serializations) | | Create Project (This Computer) -> Choose Files |
| | | import an ontology | Planned Feature | | | Planned Feature | Import -> From File | (openrefine RDF extension menu) |
| | | import from web service | | |  | | Import -> From Service | Create Project (Web Addresses) -> Choose Files |
| | | load dataset from Avro (Hadoop) | | |  | | Import -> From File | |

| | | | | | | | | |
|--|--|---------------------------------------|-------------------|---------------------------|--|---------------------------|--|--|
| | | load a dataset from database | | incanter.sql/read-dataset |  | incanter.sql/read-dataset | Import -> Database Table/Using SQL | |
| | | load multiple datasets | open-all-datasets | |  | open-all-datasets | (import several times from different sources) | Create Project (This Computer) -> Choose Files |
| | | create datasets from different data | make-dataset | to-dataset |  | make-dataset; to-dataset | (by mapping to an ontology) | GREL, Clojure, Jython + Column editing |
| | | take a subset of columns from a table | all-columns | |  | all-columns | (during the mapping process can select only several columns) | (by manually editing columns out) |

| | | | | | | | | |
|--|--|--|--|----------------------------------|---|--|---|--|
| | | concatenate datasets (by row or column) | | conj-cols; conj-rows |  | conj-cols; conj-rows | (by mapping to an ontology) | GREL, Clojure, Jython + Column editing |
| | | right join | | \$join |  | \$join | (by mapping to an ontology) | GREL, Clojure, Jython + Column editing |
| | | merge datasets by columns/rows | | conj-cols; conj-rows | | conj-cols; conj-rows | Glue Columns (drop-down menu) | GREL, Clojure, Jython + Column editing |
| | | table normalization | | melt | | melt | Glue Columns, Split Values, Fold (drop-down menu) | can be achieved, but in multiple steps |
| | | mass apply function | | grid-apply | | grid-apply | PyTransform | |
| | | filter data by value (query) | grep | \$where; query-dataset; set-data | | grep; \$where; query-dataset; set-data | PyTransform | |
| | | build a test dataset (as a subset of the input data) | test-dataset / take-rows | | | test-dataset / take-rows | (automatically done for PyTransform) | (automatically by default) |
| | | remove duplicates from dataset | Not Required as triples are idempotent | | | Not Required as triples are idempotent | | using Duplicates Facet |

| | | | | | | | | |
|--|---------|--|--|--|--|--|--|---|
| | | build a lookup table from a selection of columns | build-lookup-table | | | build-lookup-table | PyTransform | |
| | | | | | | | | |
| | columns | get a column of data (by column id) | resolve-column-id | | | resolve-column-id | (all columns available from the UI) | (all columns available from the UI) |
| | | sort data by column values | | \$order | | \$order | groupBy (for non-tabular data) | (drop-down menu) -> Sort... |
| | | apply summary function (avg, min, max, ...) to a column | | \$rollup | | \$rollup | PyTransform | GREL, Clojure, Jython |
| | | apply mathematical functions (trigonometric, arithmetic) | | (comprehensive library of all kinds of math and statistical functions) | | (comprehensive library of all kinds of math and statistical functions) | | |
| | | add a new column to the dataset (also derived) | derive-column (optional functional argument) | add-column; add-derived-column | | derive-column (optional functional argument); add-column; add-derived-column | Add Column (drop-down menu) - only empty | (drop-down menu) -> Add column based on this column.../Split into several columns |
| | | categorize column | | categorical-var; get-categories | | categorical-var; get-categories | (done automatically on import) | (drop-down menu) -> Facet -> Default facets |
| | | get column names | column-names | col-names | | column-names; col-names | (done automatically on import) | (done automatically on import) |
| | | rename column(s) | rename-columns | rename-cols | | rename-columns; rename-cols | Rename (drop-down menu) | (drop-down menu) -> Rename this column |
| | | replace column(s) | | replace-cols | | replace-cols | PyTransform | GREL, Clojure, Jython |

| | | | | | | | | |
|--|------|--|---|---------------|--|---|--------------------------------|---|
| | | reorder column(s) | swap | reorder-cols | | swap; reorder-cols | | (drop-down menu) -> Move column left/right/to beginning/to end |
| | | modify column with a custom function | mapc | transform-col | | mapc; transform-col | | GREL, Clojure, Jython |
| | | get a subset of the data by specifying columns | columns | | | columns | (by mapping to an ontology) | |
| | | using different filter for selection columnar data (column 'Facets') | | | | | | (drop-down menu) -> Facet -> Custom text or numeric facet/Customized facets |
| | | retrieve column keys that are not in the data set | invalid-column-keys | | | invalid-column-keys | PyTransform | |
| | | | | | | | | |
| | rows | extract the row with the column name (the first row) | move-first-row-to-header | | | move-first-row-to-header | (done automatically on import) | (done automatically on import) |
| | | drop rows from dataset | drop-rows, take-rows | | | drop-rows, take-rows | PyTransform | (drop-down menu for 'All') -> Edit rows -> Remove all matching rows |
| | | apply function to rows | map-rows (rows of all columns), mapc (rows of selected columns) | | | map-rows (rows of all columns), mapc (rows of selected columns) | | (by selection of the rows with GREL, Clojure, Jython) |
| | | add a row | Planned Feature | | | Planned Feature | Add Row (drop-down menu) | |

| | | | | | | | | |
|-------------------------------|-------------------|---|------|--|-----|----------------|----------------|---|
| | | get a subset of the data (by specifying rows) | rows | | | rows | PyTransform | (by selection of the rows with GREL, Clojure, Jython) |
| | | | | | | | | |
| | individual values | clustering of cells' values | | | | | | (drop-down menu) -> Edit cells -> Cluster and edit... |
| select an item from a dataset | | | sel | | sel | (by selection) | (by selection) | |

| | | | | | | | |
|----------------------|--|------------------------------------|--|--|------------------------------------|---|--|
| RDF related features | combine RDF graph from input triples | graph-fn | | | graph-fn | (not needed because ontology mapping covers all data) | (not needed because GRefine RDF skeleton covers all possible input data) |
| | create RDF triples | graph; triplify | | | graph; triplify | (drop-down menu for model) -> Publish -> RDF | (openrefine RDF extension menu) -> Edit RDF skeleton... |
| | extract subjects, predicates, objects or context out of an RDF statement | subject, predicate,object, context | | | subject, predicate,object, context | | |
| | apply prefix to a column of data | prefixer | | | prefixer | PyTransform | GREL, Clojure, Jython |
| | map data to ontology | graph-fn | | | graph-fn | UI (clicking on red circle) | (using the menu of the GRefine extension) Edit RDF Skeleton... |

| | | | | | | | |
|----------------------------------|---|---|-----|--|---|------------------------------------|--|
| | convert RDF literal into type | literal-datatype->type | | | literal-datatype->type | (not needed because of ontology) | (not needed because of ontology/rdfs) |
| | cast string to RDF Literal | (s str [lang-or-uri]) | | | (s str [lang-or-uri]) | (not needed because of ontology) | (not needed because of ontology/rdfs) |
| | specify one/more (semantic) type of a column(s) | | | | | Set Semantic Type (drop-down menu) | (using the menu of the GRefine extension) Edit RDF Skeleton... and selecting the types |
| | automatic reconciliation of entities | | | | | Extract Entities (drop-down menu) | (using the menu of the GRefine extension) Add reconciliation service... |
| | | | | | | | |
| General noteworthy functionality | undo/redo transformation(s) | N/A as grafter is a DSL - though Grafters design will enable this feature to be built easily in the Grafter GUI | N/A | | N/A as grafter is a DSL - though Grafters design will enable this feature to be built easily in the Grafter GUI | (using Command History) | (using Undo/Redo menu) |
| | map geospatial data (Google Earth) | | | | | (using a special ontology) | |
| | draw paths with geospatial data (Google Earth) | | | | | | |
| | batch mode | supported | | | supported | supported | BatchRefine + exported Undo/Redo as JSON |