

ADEQUATe

ODALIC Documentation

Version: 1.1

Date: 2017.06.01



TABLE OF CONTENTS

1	User Documentation.....	5
1.1	Introduction to Odalic.....	5
1.1.1	Odalic Semantic Table Interpretation.....	6
1.1.2	Odalic UI.....	7
1.1.3	Odalic UnifiedViews Plugin.....	7
1.2	User interface.....	7
1.2.1	Home page.....	7
1.2.2	Signing up, logging in and account management.....	8
1.2.3	New task.....	10
1.2.4	List of tasks.....	13
1.2.5	Task import.....	15
1.2.6	List of files.....	16
1.2.7	Task result.....	17
1.2.8	Knowledge base import.....	24
1.2.9	List of knowledge bases.....	25
1.2.10	New knowledge base.....	26
1.3	REST API specification.....	31
1.3.1	Content wrapping and stamps.....	31
1.3.2	HTTP error codes and headers.....	32
1.3.3	Signing up, authentication and authorization.....	33
1.3.4	Users.....	33
1.3.5	Files.....	35
1.3.6	Tasks.....	37
1.3.7	Bases.....	54
1.3.8	Advanced base types.....	55
1.3.9	Predicates and classes groups.....	56
1.3.10	Entities.....	57
1.3.11	Proposing entities.....	58
1.4	UnifiedViews DPU.....	60
1.4.1	Introduction.....	60
1.4.2	Setting up a template.....	60
1.4.3	Using the instances.....	62

1.5	Glossary.....	68
1.5.1	W3C Linked Data Glossary.....	68
1.5.2	Project glossary.....	68
2	Installation guides.....	75
2.1	Server installation.....	75
2.1.1	Prerequisites.....	75
2.1.2	Deployment.....	75
2.1.3	Building from source files.....	76
2.1.4	Security.....	76
2.1.5	Confirmation e-mails set up.....	77
2.2	Web client installation.....	77
2.2.1	Setting up a web server for the web client.....	77
2.2.2	Configuring the sign-up and password-reset confirmation address on the server.....	77
2.2.3	Configuring address of the server in the client.....	78
2.2.4	Installing LodLive application component.....	79
2.2.5	Supported browsers.....	79
2.3	Plugin installation.....	79
2.3.1	Deployment.....	79
2.3.2	Building from source files.....	80
2.4	Virtuoso installation.....	80
2.4.1	Installation.....	80
2.4.2	Datasets import.....	81
2.5	Reducing the space consumption of DBpedia.....	81
2.6	Docker image.....	82
2.6.1	Docker setup.....	82
2.6.2	Running Odalic.....	82
2.6.3	Accessing the virtual machine.....	83
2.6.4	Starting and stopping the Odalic container.....	83
2.6.5	Changing ports.....	84
2.6.6	Updating Odalic.....	84
2.6.7	Advanced Docker configuration.....	85
3	Developer Documentation.....	86

3.1	Architecture.....	87
3.1.1	Odalic UI.....	87
3.1.2	Odalic Semantic Table Interpretation.....	87
3.1.3	Odalic DPU.....	88
3.2	Server.....	88
3.2.1	Modules.....	89
3.2.2	Core algorithm description.....	91
3.2.3	REST API implementation.....	98
3.2.4	Authentication and authorization.....	99
3.2.5	Input files management and parsing.....	101
3.2.6	Tasks executions.....	101
3.2.7	User feedback.....	105
3.2.8	Result exports.....	108
3.2.9	Data cube.....	114
3.2.10	Configurations export and import.....	120
3.2.11	Persisting server state.....	123
3.2.12	Configuration.....	125
3.2.13	RDF manipulation.....	136
3.3	UI.....	137
3.3.1	Introduction.....	137
3.3.2	Architecture.....	137
3.3.3	Folder structure.....	137
3.3.4	External libraries.....	138
3.3.5	Goals.....	138
3.3.6	Application loading.....	139
3.3.7	Screens.....	140
3.3.8	Services.....	142
3.3.9	Authentication and authorization.....	143
3.3.10	Directives.....	143
3.3.11	File handling.....	145
3.3.12	Task handling.....	145
3.3.13	Taskresult screen.....	146
3.3.14	Knowledge base configuration handling.....	149
3.4	UnifiedViews DPU implementation.....	150
3.5	Possible extensions and improvements.....	150
3.5.1	Algorithm.....	150
3.5.2	UI Improvements.....	154

3.6 Project history.....154

4 Installation disc contents..... 158

5 Logos..... 159

1 USER DOCUMENTATION

- [Introduction to Odalic](#)
- [User interface](#)
- [REST API specification](#)
- [UnifiedViews DPU](#)
- [Glossary](#)

2 Introduction to Odalic

Many governments or governmental organizations throughout the world allow the public to access data they produce or collect (e.g. <http://data.gv.at/> or <http://data.opendataportal.at>). A large portion, very often data of statistical nature, of these [open data](#) is published in form of tables, encoded as common [CSV files](#). This practice, however, is not ideal, because the overall usefulness of the data would be greatly improved by making them into [Linked Open Data](#). This generally means to assign the individual pieces of content globally unique identifiers and link them to other, external sources. In detail this involves:

1. [Classifying](#) the table columns, based on their content and context against existing knowledge bases.
2. Assigning globally unique identifiers ([URIs](#), or even national characters supporting [IRIs](#)) to cell values according to Linked Data principles. Such identifiers may be reused from one of the existing knowledge bases (e.g. [DBpedia](#)).
3. Discovery of [relations](#) between columns, based on the evidence for the relations in the existing knowledge bases.
4. Converting the data in the tables and [the annotations](#) produced in the previous steps to [RDF](#); using proper data types, language tags, well-known Linked Data vocabularies (e.g. [RDF Schema](#), [DBpedia Ontology](#), ...) and other RDF-related tools and technologies.

Odalic as a platform guides its users through this process and makes it easy to reproduce, automate and customize. Based on the [work](#) of [Ziqi Zhang](#) and the [prototype implementation](#) of the described algorithm, Odalic turns it into working, user-focused application and introduces several major extensions and improvements to the original idea, while shifting the focus toward already prepared CSV files, to serve the needs of parental project [ADEQUATE](#). The original TableMiner+ algorithm has itself established as a leading solution to the problem (see

the [original paper](#) which compares it with alternative and legacy methods, distancing itself strictly from attempts to apply general NLP solutions, wrapper inducing methods and other means not tailored to work on actual tables). Development version of Odalic was a subject of one of the workshops at Semantics 2016 conference and was met with positive response and genuine interest. In cooperation with Mr. Zhang, more widespread user evaluation and feedback gathering is planned in the near future.

Odalic platform consists of **three** key **components**:

2.1.1 Odalic Semantic Table Interpretation

- It is a server, deployable to [Apache Tomcat](#) as web archive, accessible and controllable through [REST API](#) specified [here](#). This allows to draw upon its resources (as demonstrated by the [UnifiedViews plugin](#)) in new ways, unforeseen by the authors.
- Its users can provide extensive feedback on results of the automatic conversion, which our modification of the original algorithm takes into account during subsequent runs.
- Users can add their own custom resources and use them for feedback and in the exported data.
- It introduces ability to employ multiple knowledge bases at once.
- Allows export of results conforming to [CSV on the Web](#) draft specification or in popular [RDF serialization formats](#), such as Turtle and JSON+LD.
- Supports running of the conversions in independent tasks and their comfortable management.
- Supports multiple users plus administrator, employing token-based authorization and authentication friendly to further extensions.
- Includes necessary local and remote CSV files management.
- Task configuration is exportable in RDF for easier data provenance.

2.1.1.1 Summary of major implemented improvements over the original TableMiner+ algorithm

- Queries resolving the table content are now, where possible, constrained to the appropriate [types](#), thus eliminating evidently wrong results.
- The original TableMiner+ algorithm used now deprecated Freebase knowledge base. Its usage was substituted with support of DBpedia family of knowledge bases, general support for bases accessible through [SPARQL](#) endpoints and the [ADEQUATE PoolParty](#).
- All of the resources and predicates used in the code searching the bases have been externalized to configuration files and are no longer hard-coded.

- The algorithm is now more robust and efficiently handles all errors in communication with the knowledge bases.
- The algorithm now accepts constraints originating from a feedback provided by the user on top of results from previous runs. This helps to fix mistakes in automatic conversion as well as provides way to introduce custom annotations made by the users to the exported data.
- The algorithm was modified to allow computation of results forming a [data cube](#), which makes it much more usable in processing of statistical data.

2.1.2 Odalic UI

Odalic UI is a web application serving as a graphical user interface to the [Odalic Semantic Table Interpretation backend](#), allowing its users to extract and export Linked Data from provided CSV files. Its key properties are the following:

- Pleasant, easy-to-use single-page user interface.
- The data and the computed annotations are presented in the form of interactive tables, which makes it easy to overview the results, and simple to provide appropriate feedback and customize the output.
- Relations between columns are neatly visualized and modifiable in a dynamic graph component.
- Supports practically all of the server features, including separate user spaces, files and tasks management.
- Support for runtime management and configuration of proxies to the knowledge bases is provided as well.

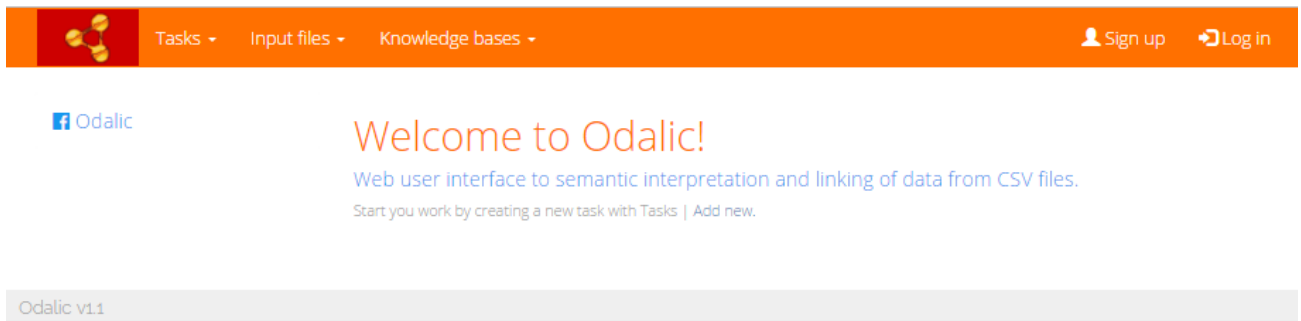
2.1.3 Odalic UnifiedViews Plugin

The plugin is built atop the [server](#) API, making it easy to run the processing within the [UnifiedViews](#). UnifiedViews is a mature [ETL](#) tool specialized on processing of and into Linked Data. Its capabilities allow to plan and schedule the use of [Odalic Semantic Table Interpretation](#) in many intricate scenarios, combining its power with other present plugins. The plugin itself manifests in the UnifiedViews as the so called [Data Processing Unit](#) (DPU), which instances can become part of arbitrarily complex virtual pipelines. The plugin otherwise follows a pattern similar to the case when the processing would be defined in the [Odalic UI](#): user specifies the input (which can now be the output of other DPUs), configuration, and connects the Odalic DPU outputs (exported table annotations or even RDF) to inputs of other DPUs. More can be found in [UnifiedViews DPU](#) user documentation.

3 User interface

3.1.1 Home page

Odalic application homepage (main menu - link of the logo):



3.1.2 Signing up, logging in and account management

Before the first use of the application you must log in. When you are not logged in, you will be automatically redirected to the login page from any other content page (Tasks, Files, etc.). When you don't have an account yet, you must sign up. **Please note that the e-mail confirmations can be turned off by the server administrator. In that case the user is logged in immediately when signing up.**

3.1.2.1 Sign up


Sign up screen

Sign up page can be opened from the main menu - item *Sign up* on the right side. You must fill your valid e-mail address, which serves as a username, and password (at least 8 characters).

The screenshot shows the sign up screen of the Odalic application. At the top, there is an orange navigation bar with a logo on the left and menu items: 'Tasks', 'Input files', and 'Knowledge bases'. On the right side of the bar are 'Sign up' and 'Log in' buttons. Below the navigation bar, the page features a yellow header with the text 'Sign up'. The form contains three input fields: 'E-mail:' with the value 'my@gmail.com', 'Password:', and 'Retype password:'. A note below the E-mail field states: '*E-mail address serves as a username.' Below the input fields is a 'Sign up' button. At the bottom, there is a link: 'Already have an account? Log in!'.


Sending of confirmation e-mail

Then you receive a confirmation message to the address you have provided.

 **E-mail sent.** An e-mail with additional instructions has been sent to the address you have provided.

Sign-up confirmation

Confirm your registration by following the link in the message.

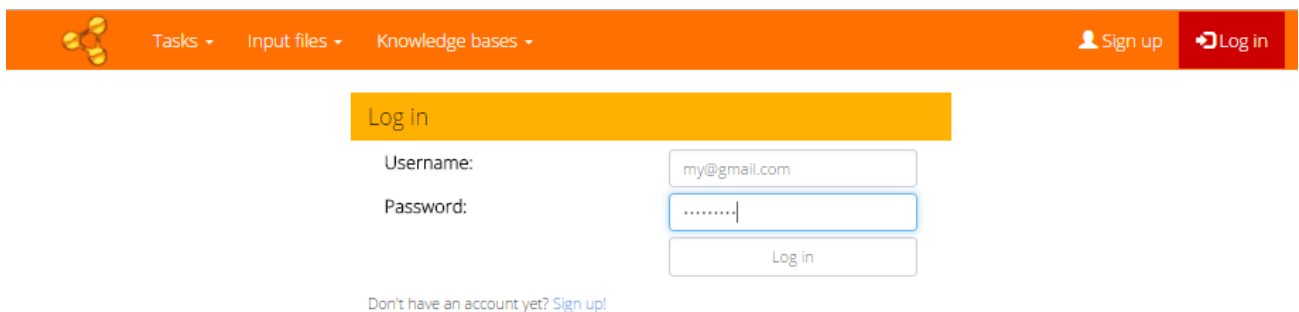
 **Welcome!** Your account has been successfully activated! Thank you for using ODALIC. [Log in now!](#)

When your account is activated, you can log in using the credentials you provided during sign-up.

3.1.2.2 Log in

Log in screen

Log in page can also be opened from the main menu (item *Log in* on the right side). Simply fill your registered e-mail and password.



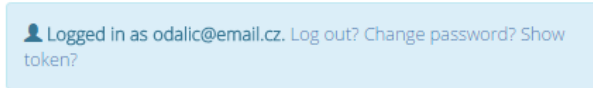
The screenshot shows the top navigation bar with a logo on the left and menu items: 'Tasks', 'Input files', and 'Knowledge bases'. On the right side of the bar are 'Sign up' and 'Log in' buttons. Below the navigation bar is a yellow 'Log in' header. The form contains two input fields: 'Username:' with the value 'my@gmail.com' and 'Password:' with a masked password '.....'. A 'Log in' button is positioned below the password field. At the bottom of the form, there is a link: 'Don't have an account yet? [Sign up!](#)'

Then the *My account* page is shown.

3.1.2.3 Account management

My account screen

When you are logged in, the item *My account* is enabled in the main menu (right side).



Password change

From the *My account* page you can also change your password. The password change must be also confirmed by the link sent to your e-mail.

The image shows a 'Change password' form. It has a yellow header bar with the text 'Change password'. Below the header, there are three rows of labels and input fields: 'Old password:' with a text box containing seven dots, 'New password:' with a text box containing seven dots, and 'Retype password:' with a text box containing seven dots. Below these fields is a note: '*Note that you have to log in afterwards.' and a 'Confirm' button.

After that, you must log in again with the new credentials.

3.1.3 New task

Page for creation of a new task configuration can be opened from the main menu - item *Tasks* > *Add new* or from the List of tasks - button *Add new* at the bottom of the page.

3.1.3.1 Basic settings

Basic task settings include Task identifier and description. Task identifier must not be empty and may contain only alphanumeric characters, spaces, dots, commas, underscores and dashes. After creation the task identifier cannot be changed, but you can always [re-import](#) the task under different identifier.



Add new task

Basic properties

Task identifier:

Description:

3.1.3.2 Setting the input file

You can select one of the already defined files, than skip to the File format configuration. Or you can add a new file. There are two options to do that:

Local files

You can upload a file from your device. The identifier will be filled automatically with the file name, but you can change it (restrictions are the same as for the task identifier).

Input file

Source: Uploaded or selected file Remote file

Upload a new file: Jugendzentren.csv

Identifier:

Selected file:

Remote files

Or you can attach a remote file. Just fill the location of the remote file and custom identifier.

Input file

Source: Uploaded or selected file Remote file

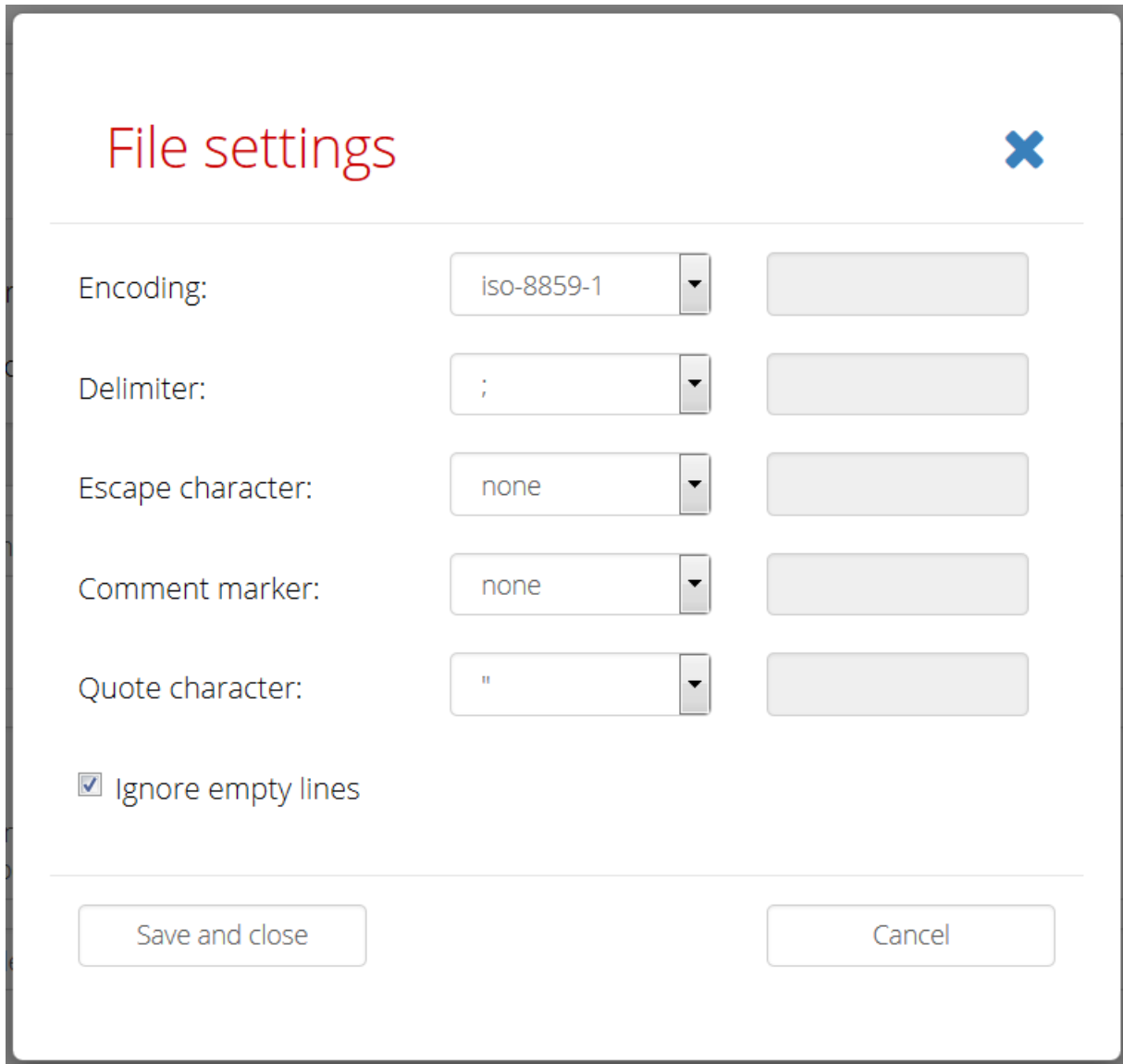
Location:

Identifier:

Selected file:

File format configuration

Select some file for processing in the task from the list of uploaded or attached files labelled *Selected file*. You can optionally configure format settings of the selected file (the dialog is opened in modal window):



The image shows a modal dialog box titled "File settings" with a red title and a blue close button (X) in the top right corner. The dialog contains several configuration options, each with a dropdown menu and a corresponding text input field:

- Encoding: dropdown set to "iso-8859-1", text input field.
- Delimiter: dropdown set to ";", text input field.
- Escape character: dropdown set to "none", text input field.
- Comment marker: dropdown set to "none", text input field.
- Quote character: dropdown set to "\"", text input field.

Below these options is a checked checkbox labeled "Ignore empty lines". At the bottom of the dialog are two buttons: "Save and close" on the left and "Cancel" on the right.

You can select some of predefined values or you can fill custom values. Custom character set names must be one of those provided in <https://docs.oracle.com/javase/8/docs/technotes/guides/intl/encoding.doc.html>. Delimiter separates records in one row. *Quote character* marks start and end of section where delimiters are ignored. *Escape character* escapes the quote character. *Comment marker* turns on comments until the end of line. You can also tell the application to *ignore empty lines*.

3.1.3.3 Knowledge bases

Select knowledge bases which will be used in the task processing. Available knowledge bases and their properties are set by configuration files. Their structure is described at [Configuration](#) part of the developer documentation. To turn them on they must be linked to from the main configuration file.

Used knowledge bases



Knowledge bases:

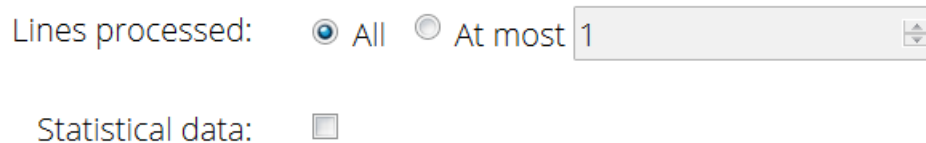
Primary knowledge base:

Select also the primary knowledge base, which is modifiable, so it will be used for proposing new entities. During export of results, the concepts from the primary knowledge base are preferred.

3.1.3.4 Processing


In the last section you can restrict the number of lines processed in the task (including the header). Also check whether you want to process the file as statistical data - in this case the relation discovery phase will be skipped, but statistical annotations will be assigned instead. They are then used when exporting results according to the [RDF data cube standard](#).

Processing



Lines processed: All At most

Statistical data:





























You can save the task, save and immediately run, or cancel the changes.

3.1.4 List of tasks

Page with list of tasks can be opened from the main menu - item *Tasks > List of tasks*.



List of tasks



Last modification	ID	Description	Used bases	Actions	State
2017-02-26 19:21	unternehmen	ODP: Top 1001 IT-Unternehmen Österreichs	ADEQUATE, DBpedia, DBpedia Clone, German DBpedia	    	
2017-02-26 09:13	states	Table of States	DBpedia, ADEQUATE, DBpedia Clone, German DBpedia	    	
2017-02-25 22:35	artist	ODP: List of artists	DBpedia, ADEQUATE, DBpedia Clone, German DBpedia	    	
2017-02-25 20:07	top-locations-wien	Top Locations in Wien	DBpedia, ADEQUATE, DBpedia Clone, German DBpedia	 	
2017-02-25 18:40	auslg_2015	Abstammung 2015 (enumeration of people)	DBpedia, ADEQUATE, DBpedia Clone, German DBpedia	    	





Add new

You can add new task by the button at the bottom of the page.






3.1.4.1 Actions

The actions that are available for each task (but can be limited according to its current state) are represented by the buttons in the Actions column.

-  ("*Play*" button) - to run the task, which will submit it to the execution service.
-  ("*Stop*" button) - to stop the task execution (only when the task is running).

-  ("*Window-arrow*" button) - to see the result of the task (only when the execution was finished).
-  ("*Down*" button) - to download the task configuration.
-  ("*Wheel*" button) - to edit the task configuration. Note that the result of the task is deleted after any substantial configuration change which might change the result.
-  ("*Bin*" button) - to delete the task. This will also unsubscribe the task from its input file, making the file eligible for deletion, if this was the last task referring to it.

3.1.4.2 States

-  (*light blue icon*) - the task is ready to be run (no result present).
-  (*dark blue icon*) - the task is running.
-  (*green icon*) - the task has finished (result exists).
-  (*yellow icon*) - the task has finished (result exists), but there were some problems (you can see the warnings if you try to display the result).
-  (*red icon*) - the task execution failed (you can see the description of error if you try to display the result).

3.1.5 Task import

Page for task import can be opened from the main menu - item *Tasks > Import*.

Task configuration can be downloaded as described on the [List of tasks](#) screen. You can import the previously downloaded task configuration from an application instance which contains files and bases that have the same name as those referred by the task. This is trivially satisfied by the the instance itself. Just fill the task identifier and upload the configuration file. Task

identifier must not be empty and may consist only of alphanumeric characters, spaces, dots, commas, underscores and dashes. After creation the task identifier cannot be changed.

Tasks ▾ Input files ▾ Knowledge bases ▾ ⚙ My account

Task import

Basic properties

Task identifier:

Configuration file

Configuration file: No file chosen

3.1.6 List of files

Page with list of files can be opened from the main menu - item *Input files* > *List of files*.



List of input files




Created	File name	Actions	Type
2017-02-13 12:10	Jugendzentren.csv	  	
2017-02-12 17:30	LT-2009.csv	  	
2017-02-13 12:49	SCHULEOGD.csv	  	
2017-02-12 15:22	STMK_2012_LIVEBIRTHS.csv	  	
2017-02-13 13:14	artist.csv	  	

First Previous **1** 2 Next Last



Add new

You can add a new file during creation of a new task (as described in [New task](#)). File can be added also by the button at the bottom of the page or from the main menu - item *Input files* > *Add new*.

3.1.6.1 Actions

-  ("Down" button) - to download the file.
-  ("Wheel" button) - to configure format settings of the file (dialog is opened in modal window). See [New task](#) for the fields description.
-  ("Bin" button) - to delete the file. Not that only files that are not referred by any task can be deleted.

3.1.6.2 Types

-  ("Pin" icon) - indicates uploaded (local) file.
-  ("Earth" icon) - indicates remote file. When the content of the file changes, application will not register that and re-run of the task will produce different results.

3.1.7 Task result

Page with the result of semantic annotation can be opened from the [List of tasks](#) by the "Window-arrow" button in the *Action* column of the table with tasks. Here you can see the result of classifications, disambiguations, subject column detections and relation discovery process, adjust feedback for the algorithm and export resulting annotations in the form of extended CSV file, JSON annotations and RDF triples.

3.1.7.1 Classifications and disambiguations

In the first tab you can review resources suggested as classifications of the columns and disambiguations of the cells. Resources from different knowledge bases are distinguished by colours and made bold for the primary knowledge base. The name of corresponding knowledge base is shown in a tooltip (when the mouse is over) of the resource label. The link next to the resource label opens a resource web page (in a browser's new tab).

Missing resource annotation can be caused by several reasons: the way how the algorithm viewed the processing of column is shown in the tooltip (when the mouse is over) of the column header text. There are four possible ways the algorithm can view a column:

- **Named entity** - the algorithm recognized the values of column as named entities (uniquely identifiable by an URI from the base) and performed classification and disambiguation process normally.
- **Non-named entity** - the algorithm assessed that the values are not named entities (for example when there are numbers), so the classification and disambiguation process was skipped for that column.
- **Ignored** - the column was intentionally ignored, because the user asked so in the feedback to previous task run.
- **Compulsory** - the column was classified and disambiguated although it was not recognized as named entity, because the user asked so in the feedback to previous task run.

Tasks - Input files - Knowledge bases - My account

Classifications and disambiguations Subject columns Relations

Column classifications and disambiguations of the values in cells

Ort	PLZ	Bezirk	Bundesland
settlement (dbpedia-owl:Settlement) settlement (dbpedia-owl:Settlement) Spatial Thing (geo:SpatialThing) Concept (skos:Concept)	population total (dbpedia-owl:populationTotal) PLZ (dbpprop:plz) PLZ (http://de.dbpedia.org/property/plz)	Q 486972 (http://www.wikidata.org/entity/Q486972) Item (http://www.wikidata.org/ontology#item) Place (schema-org:Place) Concept (skos:Concept)	Q 486972 (http://www.wikidata.org/entity/Q486972) Region (yago:Region108630985) Place (schema-org:Place) NUTS Region (http://ec.europa.eu/eurostat/ramon/
Linz (dbpedia:Linz) Linz (dbpedia:Linz) Linz (http://de.dbpedia.org/resource/Linz) J. J. Linz (http://linked.opendata.cz/ontology/domain/vavai/riv/klicoveSlovo/J.%20Linz)	4040	Linz-Stadt Stadt Linz@de (http://www.wikidata.org/entity/Q15848788) Kinderheim der Stadt Linz@de (http://de.dbpedia.org/resource/Liste_von_Kinderheimen_in_Österreich)	Oberösterreich Oberösterreich (dbpedia:Upper_Austr) Oberösterreich (dbpedia:Upper_Austr) Oberösterreich (http://de.dbpedia.org/AT31 - Oberösterreich (http://ec.europa.eu/eurostat/ramon/
Linz (dbpedia:Linz) Linz (dbpedia:Linz) Linz (http://de.dbpedia.org/resource/Linz) J. J. Linz (http://linked.opendata.cz/ontology/domain/vavai/riv/klicoveSlovo/J.%20Linz)	4020	Linz-Stadt Stadt Linz@de (http://www.wikidata.org/entity/Q15848788) Kinderheim der Stadt Linz@de (http://de.dbpedia.org/resource/Liste_von_Kinderheimen_in_Österreich)	Oberösterreich Oberösterreich (dbpedia:Upper_Austr) Oberösterreich (dbpedia:Upper_Austr) Oberösterreich (http://de.dbpedia.org/AT31 - Oberösterreich (http://ec.europa.eu/eurostat/ramon/
Leonding (dbpedia:Leonding) Leonding (dbpedia:Leonding) Leonding (http://de.dbpedia.org/resource/Leonding)	4060	Linz-Land Linz-Land District@en (dbpedia:Linz-Land_District) Bezirk Linz-Land@de (http://www.wikidata.org/entity/Q265632) Bezirk Linz-Land@de (http://de.dbpedia.org/resource/Bezirk_Linz-Land)	Oberösterreich Oberösterreich (dbpedia:Upper_Austr) Oberösterreich (dbpedia:Upper_Austr) Oberösterreich (http://de.dbpedia.org/AT31 - Oberösterreich (http://ec.europa.eu/eurostat/ramon/
Gallneukirchen (dbpedia:Gallneukirchen) Gallneukirchen (dbpedia:Gallneukirchen) Gallneukirchen (http://de.dbpedia.org/resource/Gallneukirchen)	4210	Urfahr-Umgebung Urfahr-Umgebung District@en (dbpedia:Urfahr-Umgebung_District) Bezirk Urfahr-Umgebung@bar (http://www.wikidata.org/entity/Q253585) Bezirk Urfahr-Umgebung@de (http://de.dbpedia.org/resource/Bezirk_Urfahr-Umgebung)	Oberösterreich Oberösterreich (dbpedia:Upper_Austr) Oberösterreich (dbpedia:Upper_Austr) Oberösterreich (http://de.dbpedia.org/AT31 - Oberösterreich (http://ec.europa.eu/eurostat/ramon/
Wels (dbpedia:Wels) Wels (dbpedia:Wels) Wels (http://de.dbpedia.org/resource/Wels_(Stadt)) Wels (http://linked.opendata.cz/ontology/domain/vavai/riv/klicoveSlovo/wels)	4600	Wels-Stadt Kategorie:Wels (Stadt)@de (http://www.wikidata.org/entity/Q7372208) Wels (Stadt)@de (http://de.dbpedia.org/resource/Wels_(Stadt))	Oberösterreich Oberösterreich (dbpedia:Upper_Austr) Oberösterreich (dbpedia:Upper_Austr) Oberösterreich (http://de.dbpedia.org/AT31 - Oberösterreich (http://ec.europa.eu/eurostat/ramon/

First Previous 1 2 Next Last

Reexecute Save Cancel

Related feedback

You can adjust basic feedback settings directly from the table of results. The lock icon shows whether the feedback is set for the particular cell or column header. If so it should be reflected by the algorithm in the next run of the task. Initially it is unlocked. When you adjust some feedback, it locks automatically. You can also lock and unlock it manually.

- **"Bin" icon** - deletes resource annotation for particular knowledge base. For cells it means that the disambiguation of the cell will be skipped, for column headers it means that the whole column will be skipped (ignored).
- **"Plus" icon** - opens dialog for proposing a new resource, which will be then used for new disambiguation of the cell (or classification of the column). It is possible to propose resources only to the primary knowledge base (which must be modifiable).

Propose disambiguation ✕

Label *

Alternative label

Alternative label



Custom URL suffix

Class:

- "Arrow" icon - opens a dialog with detail of classification/disambiguation (described bellow).

Cell disambiguation

DBpedia (Primary knowledge base)

  Oberösterreich 2.05 ×
(dbpedia:Upper_Austria)



▼ Search DBpedia

Search

Use



Propose

ADEQUATe

  Oberösterreich 2.16 ×
(http://adequate-project-pp.semantic-web.at/ADEQUATe_KB/009d929f-c82d-40c2-80da...



▼ Search ADEQUATe

DBpedia Clone

  Oberösterreich 2.19 ×
(dbpedia:Upper_Austria)

▼ Search DBpedia Clone

German DBpedia

  Oberösterreich 2.16 ×
(http://de.dbpedia.org/resource/Oberösterreich)

▼ Search German DBpedia

Skip disambiguation of the cell

The detail dialog shows annotations found for cell/column in their entirety. It offers more options for feedback adjustment than the table perspective.

The smart select box displays chosen annotation for each knowledge base. When you click to the select box next to the chosen annotation, other candidates suggested by the algorithm are

displayed and you can select one of them as the new annotation resource. Every item in the select box shows label, URI of the resource and a score computed by the algorithm. The available commands in the smart select box are the following:

- By clicking the **"i" icon** you can open the resource web page (in the new tab, if the underlying base has such HTML representation published).
- The **"chain" icon** opens the LodLive component with the corresponding resource.
- The **"cross" icon** deletes the annotation which means that the cell / column will be skipped in the next run.

When you want to set another resource (different from the candidates provided by algorithm), you can search it in the associated knowledge base (by the label). Then select one of the results and *Use* it as the annotation. For the primary knowledge base you can also propose completely new resource (as described above). By the last checkbox you can completely skip the cell/ignore column (for all knowledge bases). The lock icon works in the same way as in the result table.

3.1.7.2 Subject columns

In the second tab (available only for a task which does not process statistical data - see [Task configuration](#)) you can review suggested subject columns. Subject columns serve as sources for relation discovery (depicted in the third tab). The subject column suggested by algorithm is emphasized in the table preview for each knowledge base. Grey marked column shows an algorithm suggestion (or user feedback from previous run). When we want to mark another columns as subject columns, just click on the desired columns and they will be emphasized by orange color. Adjusting of this part of the feedback is also confirmed by the lock icon which works in a similar way as in the previous tabs.

3.1.7.3

3.1.7.4 Relations

In the third tab (also only for task which does not process statistical data) you can review suggested relations between the subject column and other columns. Relations are depicted in the directed graph, where vertices represent columns and relations are represented by edges labelled with predicate resources found in the knowledge bases. Click on the edge label opens dialog with details similar to those for classifications or disambiguations in the first tab. Lock icons work also in the same way.

Tasks ▾ Input files ▾ Knowledge bases ▾ My account

Classifications and disambiguations Subject columns **Relations**

Relations between columns

Node dragging Link creation Reset layout

Reexecute Save Cancel

Export

Warning! Export may not correspond to the changes you have made. A task reexecution may be needed.

Extended CSV Annotations RDF (Turtle) RDF (SON-LD)





















First mode of the graph, called *Node dragging*, allows dragging of the vertices around the screen, thus changing the layout. When you switch to the second mode by the *Link creation* button, then you can add new edges representing new relations between some columns. After their creation it also opens the dialog with details, where you can search or propose the new predicate resource annotation.

3.1.7.5 Tab for statistical data processing

When you set the statistical data processing check box to turned on state in the [Task configuration](#), the subject column detection and relation discovery parts of the algorithm are skipped. The algorithm instead prepares statistical annotations for a data cube. This means that the algorithm will determine which columns are dimensions and which one measures. Initially all named entity columns are considered for dimensions and non-named entity columns for measures.

Statistical data cube set-up

















Dimensions

Column	Predicate	Action
NUTS1		   
NUTS2		   
NUTS3		   
DISTRICT_NAME		   
LAU_NAME		   

 ▾

Add to dimensions

Measures

Column	Predicate	Action
DISTRICT_CODE		   
LAU_CODE		   
LIVEBIRTHS		   
YEAR		   

 ▾

Add to measures

Reexecute Save Cancel

You can eventually change this decision. **"Bin" icon** removes a column from particular group, **"Add to ..."** buttons add selected column to the group.

After that you should set the predicate resource in a way similar to classification or disambiguation. "Plus" icon proposes new resource, lock icon confirms the settings and **"Arrow" icon** opens the dialog with details, where you can search resources in the knowledge base. Predicate resources are set only for the primary knowledge base. The predicates are eventually used in the next run of the algorithm (if you hit *Reexecute*) for creating the so-called "Observations" according to the [data cube vocabulary](#).

3.1.7.6 Reexecute

To consider changes which were provided as a feedback, the algorithm must be reexecuted. You can reexecute the task directly by the **"Reexecute" button**. Or you can just save the provided feedback by the **"Save" button** and start the execution later.

3.1.7.7 Export

At any time you can export the results by buttons at the end of the Result page. Only the result of the last finished execution is used as basis for the exports (thus ignoring a feedback that was not baked in by the task re-execution).

Export

Extended CSV

Annotations

RDF (Turtle)

RDF (JSON-LD)

- Extended CSV - file containing columns of the task input file plus extra columns with annotations (disambiguation resources) . The extra columns are easily distinguishable from the original ones, because they are named **_url* where *** stands for the original column name.
- Annotations - annotations for columns of the extended CSV file according to the [CSV on the Web](#) standard (including annotations of virtual columns used for classifications, relations or statistical data cube).
- RDF - linked data triples in form of RDF describing disambiguated values with classification and relations (or data cube observations in case of statistical data processing).
 - There are two common RDF formats available: [Turtle](#) and [JSON-LD](#).

3.1.8 Knowledge base import

Page for knowledge base import can be opened from the main menu - item *Knowledge bases > Import*.

Knowledge base configuration can be downloaded as described on the [List of knowledge bases](#) screen. Just fill the knowledge base identifier and upload the configuration file. Knowledge base identifier must not be empty and may consist only of alphanumeric characters, spaces, dots, commas, underscores and dashes. After creation the knowledge base identifier cannot be changed.

Knowledge base proxy configuration import

Basic properties

Name:

Configuration file

Configuration file: No file chosen

3.1.9 List of knowledge bases

Page with list of knowledge bases can be opened from the main menu - item *Knowledge bases* > *List of knowledgebases*.

List of knowledge bases

Name	Description	Actions
DBpedia	DBpedia	<input type="button" value="Download"/> <input type="button" value="Settings"/> <input type="button" value="Copy"/> <input type="button" value="Delete"/>
DBpedia Clone	DBpedia Clone	<input type="button" value="Download"/> <input type="button" value="Settings"/> <input type="button" value="Copy"/> <input type="button" value="Delete"/>
German DBpedia	German DBpedia	<input type="button" value="Download"/> <input type="button" value="Settings"/> <input type="button" value="Copy"/> <input type="button" value="Delete"/>
OpenData	OpenData	<input type="button" value="Download"/> <input type="button" value="Settings"/> <input type="button" value="Copy"/> <input type="button" value="Delete"/>

First Previous 1 Next Last

New knowledge base can be added by the button at the bottom of the page or from the main menu - item Knowledge bases > Add new . It can be also imported from configuration file (import button at the bottom of the page or he main menu - item Knowledge bases > Add

new) or user can derivate a new knowledge base from the already inserted knowledge base by the "clone" button in the column Actions.

3.1.9.1 Actions



("Down" button) - to download the knowledge base configuration.



("Wheel" button) - to configure configuration of the knowledge base (dialog is opened in modal window). See [New knowledge base](#) for the fields description.



("Clone" button) - to derivate a new knowledge from the already inserted knowledge base. This action shows the pre-filled form based on the selected knowledge base for the creation the new knowledge base.



("Bin" button) - to delete the knowledge base. Only knowledge bases that are not referred by any task can be deleted.

3.1.10 New knowledge base

Page for creation of a configuration to a new knowledge base proxy can be opened from the main menu - item *Knowledge bases* > *Add new* or from the List of knowledge bases (button *Add new* at the bottom of the page or "clone" button in the column Actions) .

The configuration of a new knowledge base is divided into four parts - *Main*, *Search*, *Propose* and *Advance*.

3.1.10.1 Main

Main knowledge base settings include the required knowledge base identifier, end-point URL and an optional description. The knowledge base identifier must not be empty and may contain only alphanumeric characters, spaces, dots, commas, underscores and dashes. After creation the knowledge base identifier cannot be changed, but you can always [re-import](#) the

knowledge base under different identifier. *End-point URL* must be a valid URL in the standard format referring to an available knowledge base.

Tasks ▾ Input files ▾ Knowledge bases ▾ My account

Knowledge base proxy configuration

Main Search Propose Advanced

Identifier: ⓘ* DBpedia Clone

Description: DBpedia Clone

End-point URL: ⓘ* http://dbpedia.org/sparql Open

Save Cancel

3.1.10.2 Search

In search part you can set parameters for searching user entities or properties, such as type of search, preferred or useful classes and prefixes.

Text searching method offers three types of searching - *fulltext*, *exact match* or *substring*. Search times depend on the chosen type. Also the fulltext might not be supported by the chosen base, choose other methods in that case.

In *Language tag* you can be set language shortcut for searching. Usually we choose the most common in the chosen knowledge base.

Skipped attributes offer a modifiable list of attributes that are skipped during property searching because they are inappropriate for some reason (too general, resulting in unwanted outputs,...).

Skipped classes offer a modifiable list of classes that are skipped when entity searching because they are inappropriate for some reason too.



Knowledge base proxy configuration

Main Search **Propose** Advanced

Text searching method:

Language tag:

Skipped attributes:







Skipped classes:



3.1.10.3 Used groups

What follows is the list of available groups (some of them are part of the default installation, but other can be added by hitting the Add group button below the table). Groups are used to describe the structure of the base, mainly what predicates are used indicate instance-of-relations, what types are used and so on. These are vital for proper behaviour of the algorithm. When the *Autodetected* option is on, the application attempts to deduce the used groups before every task run by querying the base. The user can override this behaviour and choose the groups manually by un-checking the *Autodetected* check-box and checking the desired groups in the table. The groups can be further edited or even removed using the associated buttons.

Used predicates and classes groups

Autodetection: 

Used	Name	Actions
<input type="checkbox"/>	dbpedia	 
<input checked="" type="checkbox"/>	dc	 
<input type="checkbox"/>	de.dbpedia	 
<input checked="" type="checkbox"/>	de.dbpedia.owl	 
<input type="checkbox"/>	de.dbpedia.rdf	 

- 
("Wheel" button) - to edit the group properties.
- 
("Bin" button) - to delete the group properties.

Group properties

In the group properties the user can set URIs for the predicates and types that form a cohesive set used in the common bases to denote their structure. it is not necessary to have all the subgroups non-empty, but overall at least one of the used groups must have at least the *instance-of* and *label* predicates set.



Group properties

Name: ⓘ	<input type="text" value="de.dbpedia.rdf"/>	
"label" predicates: ⓘ	<input type="text" value="http://www.w3.org/2000/01/rdf-schema#label"/>	<input type="button" value="Remove"/>
	<input type="text"/>	<input type="button" value="Add"/>
"description" predicates: ⓘ	<input type="text"/>	<input type="button" value="Remove"/>
	<input type="text"/>	<input type="button" value="Add"/>
"instance of" predicates: ⓘ	<input type="text" value="https://www.w3.org/1999/02/22-rdf-syntax-ns#type"/>	<input type="button" value="Remove"/>
	<input type="text"/>	<input type="button" value="Add"/>
Class types: ⓘ	<input type="text"/>	<input type="button" value="Remove"/>
	<input type="text"/>	<input type="button" value="Add"/>
Property types: ⓘ	<input type="text" value="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>	<input type="button" value="Remove"/>
	<input type="text"/>	<input type="button" value="Add"/>

3.1.10.4 Propose

In the propose section the user can set several properties if he wants to create a primary knowledge base proxy configuration. This means the user can insert own proposals for entities or properties into this modifiable knowledge base, also it can be used to break ties in conflicts.

If *Insert enabled* check-box is marked, it shows the expanded form depicted below.

Insert end-point URL must be a valid URL in the standard format referring to an available knowledge base endpoint dedicated to modifying the base. If this field is empty the algorithm uses end-point URL from the previous step *Main*.

Insert to graph keeps the graph where the new proposal will be saved. If this field is empty, the algorithm uses default graph.

User classes prefix sets the prefix of URL for the user proposal of class or a property and *User resource prefix* sets the prefix of URL for the user proposal that is not a class or property.

Object property type and *datatype property type* allow to configure the types used to denote the two kinds of properties, where the first one relates individuals resource, while the latter one an individual and a literal of some data type, e.g. string.

Tasks ▾ Input files ▾ Knowledge bases ▾ My account

Knowledge base proxy configuration

Main Search **Propose** Advanced

Insert enabled:

Insert end-point URL:

Insert to graph:

User classes prefix:

User resource prefix:

Object property type:

Datatype property type:

Save Cancel

3.1.10.5 Advanced

In the advanced section the user can add some special key-values for new knowledge bases if they support them (for example some tuning options, additional authorization attributes) and he or she can provide authorization values to access bases protected by basic authorization according to the *Type*. So far only the SPARQL-based bases are supported (but the developers are free to add other). The interpretation of the keys and their values depends on the chosen type. The provided SPARQL type implementation neither requires nor supports any.



Knowledge base proxy configuration

Main

Search

Propose

Advanced

Type: ⓘ

SPARQL

Additional properties: ⓘ

Key	Value	Comment	Actions
<input type="text"/>	<input type="text"/>		

Add row

Authorization

Login: ⓘ

Password: ⓘ

Save

Cancel

4 REST API specification

4.1.1 Content wrapping and stamps

All regular API responses (which are of media type *application/json*) share the same format:

```
{ status: XXX, type: "YYY", payload: ZZZ, stamp:
"client_defined_string" }
```

The `status` contains an integer, the same as the code of the HTTP response itself. The `type` is either `DATA` or `MESSAGE` string. When the type is `DATA`, the `payload` contains a serialized domain object. When the type is `MESSAGE`, then the `payload` has the following standard format:

```
{ text: "AAA", additionalResources: [ "BBB", "CCC", ... ],
debugContent: "DDD" }
```

The message itself is stored in the `text`, which may contain confirmation message or a message associated with an exception, depending on the status code of the response. The `additionalResources` is an array of URI strings, that may provide the client programmer with additional resources (usually documentation) to help him or her understand the message or debug its causes. The `debugContent` holds additional piece of information essential in debugging the call (usually stack trace) when the application server is in debug mode. The `text` and `debugContent` can be null (or non-existent in the message), although in case of the `text` this is not recommended. The `additionalResources` cannot be null, but an empty array at most, when no additional resources can be provided. The wrapping was introduced as a mean of making the interpretation of the response simpler for the client. The client can now determine the basic flow of processing of the response without the need to parse the payload.

The `stamp` field contains a string, that was sent to server by client in the request that originated the sent response. Clients can add an optional query parameter named `stamp` (e.g. `&stamp=client-set-string`) with a value of their choosing to every request URL, and the server will send it back to them as a part of the response wrapping object.

To save space the following examples present data payloads in an unwrapped form, in which case the type is assumed to be `DATA` and the status code is as specified for the HTTP response. When the client sends data to server, the wrapping is naturally not used. Also when the returned media type is not an *application/json* or when the returned content is a result of explicit export (such is the case of the annotated table), the response is also not wrapped and in case of error only the error status is set (as HTTP header) and no wrapped JSON message is sent. Unless specified otherwise, the server can leave fields with null values from the result out.

The following examples use a `http://odalic.eu/odalic/` as the API calls URL prefix, where `./odalic` is the API root. Strings enclosed with `{` and `}` stand for custom identifiers. All calls unless said otherwise require [Authentication and authorization](#) via Authorization header with its value set to "Bearer <issued token>".

4.1.2 HTTP error codes and headers

The API follows standard conventions and returns appropriate HTTP status codes. The 500 should be returned only when the server itself fails. Less common codes will be described in individual cases. The server respects and in some cases (export, input download) even discriminates media types set in Accept and Content-Type headers.

4.1.3 Signing up, authentication and authorization

- All API calls apart from those that allow to sign up, confirm registration or change password, require token authentication from the client as introduced in MISSING LINK .
 - The authentication is done by passing a token issued by the server for a client in the standard *Authorization* header. It must start with **Bearer** <token>, where **Bearer** is a constant string (a standard way of telling the server which authentication scheme is used) separated from the token by a single space.
- User signs up by providing an email address and password and following the confirmation link sent to the provided address.
- User can set a new password by providing the old one and new one and following the confirmation link sent to the provided address. This also invalidates issued authentication tokens.
- Confirmation and authentication tokens have limited expiration time, set in the server configuration at `config/sti.properties`.
- All registered users share the same role of a common user. Apart from them a singular administrator account is created based on the details provided in the `config/sti.properties`. This account has all the capabilities of a regular user, but it can also list all the users and impersonate them by adding "users/{userId}" at the beginning of the request path. These URLs are valid for the regular users too, but are unnecessary, because the server has the user already authenticated and accessing other users' resources results in authorization errors.
- Note that the signing up, authentication and authorization features require proper server configuration of the mail service, administrator account, appropriate confirmation links and other!
- As a result of this, all tasks and files (**but not knowledge bases or their entities**) live in separate name spaces and users can only access those which they own. This also means

that two users can access the same URL (e.g. when they both named their files the same), but its content will differ. This also helps to maintain partial backward compatibility for the previous version of the API, which assumed a single user.

4.1.4 Users

- (1) **POST** `http://odalic.eu/odalic/users/`
- (2) **GET** `http://odalic.eu/odalic/users/` (only for the administrator)
- (3) **POST** `http://odalic.eu/odalic/users/confirmations`
- (4) **POST** `http://odalic.eu/odalic/users/authentications`
- (5) **GET** `http://odalic.eu/odalic/users/{user@odalic.eu}`
- (6) **PUT** `http://odalic.eu/odalic/users/{user@odalic.eu}/password`
- (7) **POST** `http://odalic.eu/odalic/users/passwords/confirmations`
- (8) **DELETE** `http://odalic.eu/odalic/users/{user@odalic.eu}` (only for the administrator)
- Query (1) signs the user up using the user's chosen credentials (valid e-mail and password; see (US1) for example).
 - This call does not require token authentication.
 - Although the user is signed up, its calls are not authorized until the provided e-mail address is confirmed via the link sent to his or her email. See (3).
- Query (2) allows the administrator to list the active users. It returns array of (US3).
 - Administrator's authentication works the same way as for the regular users, see (4).
- Query (3) accepts a confirmation token (see (US2)). When it is valid and identifies a signed-up user, its account is activated.
 - This call does not require token authentication.
 - Only active user are able to authenticate and get the access token issued, see (4).
 - This call is also initiated by a web client, when its user arrives at the address specified in `config/sti.properties` that was sent within the confirmation e-mail.
- Query (4) accepts user credentials (the same as (1), see (US1)) and validates them against the set of active users. When OK, access token is issued as a response (same format as (US2)).

- This call does not require token authentication.
- Query (5) only returns the calling user e-mail and role as demonstrated in (US3).
- Query (6) allows the user to change the password. It accepts (US4), which contains the old and new one. The old one is used to authenticate the user.
 - This call does not require token authentication.
 - The change does not take effect until the confirmation token is provided, see (7).
- Query (7) confirms the setting of the new password. It accepts confirmation token (US2) whose value was sent in the e-mail (the same process as for the user activation).
 - This call does not require token authentication.
 - When the password is changed, all previously issued authentication tokens are invalidated and their use will result in authentication error.
- Query (8) un schedules all the user's tasks, deletes his or her tasks and files and finally deletes the user's entry, thus prevent him or her to log in.

4.1.4.1 (US1)

```
{ "email": "test@odalic.eu ", "password": "3*(ew4zpor8ad 89z /*09892*(^Y&" }
```

(US2)

```
{ "token": "eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJzaWduZXIiLCJpc3MiOiJJPZGFsaWwMiLCJleHAiOiJlODU3NTQwMzAsImp0aSI6IjkyOTU2MjUyLTk2ZTMtNGE0MC05ZWE3LTlmYzQ1MmMlMTkxMyJ9.d0tIzt8O_P-QhGZHswBF-KbP9v8V1wscLK5Cph1i_a0" }
```

(US3)

```
{ "email": "test@odalic.eu", "role" : "USER" }
```

(US4)

```
{ "oldPassword": "3*(ew4zpor8ad 89z /*09892*(^Y&", "newPassword" : "much easier password" }
```

4.1.5 Files

- (1) **PUT** `http://odalic.eu/odalic/files/{file_name}`
- (2) **GET** `http://odalic.eu/odalic/files/{file_name}`
- (3) **DELETE** `http://odalic.eu/odalic/files/{file_name}`
- (4) **GET** `http://odalic.eu/odalic/files/`
- (5) **PUT** `http://odalic.eu/odalic/files/{file_name}/format`
- (6) **GET** `http://odalic.eu/odalic/files/{file_name}/format`
- Query (1) uploads a file description (FS1) of a remote file or a cached local file, depending on the accept type (in which case (FS1) does not apply, only the file itself is uploaded)
 - Identifier 'file_name' must be unique. Please note that the identifier can contain only alphanumeric characters, underscore, dash, space, comma - all must be URL encoded.
 - If the MIME type produced by the client is *multipart/form-data*, then the server expects only the file itself in the payload under *input* part label.
 - If the MIME type produced by the client is *application/json*, then the server expects only the remote file description containing and assumes that the location is specified by the location attribute in the sent JSON object. The location can contain any valid URL.
 - The upload is non-resumable.
 - There is also available an alternative **POST** API call at `../files` URL for the *multipart/form-data* version, which derives the ID from the name of the uploaded file.
 - The format field is optional and allows for a remote file to immediately specify its format. Only the fields in the example (whose names are self-explanatory) are supported. The `quoteCharacter`, `escapeCharacter`, `commentMarker` can be set to null or omitted (in that case a default parser value is set or the parsing is automatic in that aspect). For a file that has been uploaded via *multipart/form-data*, a default format (shown in the (FS1) example) is set and must be configured (if not suitable for the data) by a separate subsequent API call (see (5), (6)). The `charset` field value is expected to be a canonical name recognized by Java NIO (first column at <https://docs.oracle.com/javase/8/docs/technotes/guides/intl/encoding.doc.html>).
- Query (2) returns either a file description (FS2) for file with id '**file_name**' or the file itself, depending on the MIME type set by the client.
 - If the MIME type is set to *application/json*, then the file description object is returned.

- If the MIME type is set to *text/csv*, then the file specified in the location is provided. In case of remote location, the server attempts to download the file from the location and passes it as its own.
- See (FS2) for output format of the file description. Notice the `cached` attribute that distinguishes uploaded (cached) files from the remote ones.
- Query (3) deletes the file (if local) and its corresponding file description.
 - Returns HTTP 406 Conflict and an error message containing the IDs of the referring tasks, when the file is still utilized in some task configuration.
- Query (4) returns a list of all the files.
 - See (FS3) below for further details on the returned data.
- Query (5), (6)
 - Only the fields in the payload example (FS4) (whose names are self-explanatory) are supported. The `quoteCharacter`, `escapeCharacter`, `commentMarker` can be set to null or omitted when put (in that case a default parser value is set or the parsing is automatic in that aspect). The `charset` field value is expected to be a canonical name recognized by Java NIO (first column at <https://docs.oracle.com/javase/8/docs/technotes/guides/intl/encoding.doc.html>).

4.1.5.1 (FS1)

```
{ "location": "http://odalic.eu/odalic/files/file_name", "format":
  { "charset" : "UTF-8", "delimiter" : ";", "emptyLinesIgnored" :
true, "quoteCharacter" : null, "escapeCharacter" : null,
"commentMarker" : null } }
```

4.1.5.2 (FS2)

```
{ "id": "file_name", "uploaded": "yyyy-MM-dd HH:mm", "owner":
"no_meaning_in_this_version_just_some_string", "location":
"http://odalic.eu/odalic/files/file_name", "format": { "charset" :
"UTF-8", "delimiter" : ";", "emptyLinesIgnored" : true, },
"cached": false }
```

4.1.5.3 (FS3)

```
[ { "id": "file_name", "uploaded": "yyyy-MM-dd HH:mm", "owner":
```



```
"no_meaning_in_this_version_just_some_string", "location":  
"http://odalic.eu/odalic/files/file_name", "format": { "charset" :  
"UTF-8", "delimiter" : ";", "emptyLinesIgnored" : true }, "cached":  
false }, ... ]
```

4.1.5.4 (FS4)

```
{ "charset" : "windows-1250", "delimiter" : ",",  
"emptyLinesIgnored" : true, "quoteCharacter" : "\"",  
"escapeCharacter" : "\\\"", "commentMarker" : "#" }
```

4.1.6 Tasks

- (1) **PUT** `http://odalic.eu/odalic/tasks/{task_id}`
- (2) **GET** `http://odalic.eu/odalic/tasks/{task_id}`
- (3) **DELETE** `http://odalic.eu/odalic/tasks/{task_id}`
- (4) **PUT** `http://odalic.eu/odalic/tasks/{task_id}/configuration`
- (5) **GET** `http://odalic.eu/odalic/tasks/{task_id}/configuration`
- (6) **PUT** `http://odalic.eu/odalic/tasks/{task_id}/configuration/feedback`
- (7) **GET** `http://odalic.eu/odalic/tasks/{task_id}/configuration/feedback`
- (8) **GET** `http://odalic.eu/odalic/tasks/{task_id}/configuration/feedback/input`
- (9) **PUT** `http://odalic.eu/odalic/tasks/{task_id}/execution`
- (10) **DELETE** `http://odalic.eu/odalic/tasks/{task_id}/execution`
- (11) **GET** `http://odalic.eu/odalic/tasks/{task_id}/result`
- (12) **GET** `http://odalic.eu/odalic/tasks/result/rdf-export`
- (13) **GET** `http://odalic.eu/odalic/tasks/{task_id}/state`
- (14) **GET** `http://odalic.eu/odalic/tasks/?states={false, true}&orderBy={id,created}`
- Query (1) creates a new task definition with the identifier 'task_id'.
 - See (TS1) for the task description format. The id field is optional, but when filled, then it must be equal to the id provided in the URL.
 - Also consumes *text/turtle* (must be set as the Content-Type of the request) in the format generated by (2) when *text/turtle* requested.
- Query (2) returns the task definition (TS1).
 - Also produces *text/turtle* when the Accept header is set to *text/turtle*.

- Query (3) removes the task. Although the task is deleted, any running computation might not be stopped immediately.
- Query (4) sets a new task configuration for possible next execution.
 - See (TS2) for the format description. The primary base has to always be set to one that is modifiable (see Bases).
 - The `rowsLimit` field when set to something else than null cuts of the input after the specified number of rows (the `rowsLimit` must be non-negative when set, and less than or equal to $2^{31} - 1$, which is also the maximum supported number of rows for the files). The uploaded files remain intact.
- Query (5) returns the task configuration object (TS2).
 - The `usedBases` field must be a subset of the available bases. The `primaryBase` must be one of the used and modifiable ones. The `usedBases` can be left out or set to null, in which case it defaults to all available bases.
- Query (6) sets a feedback that the algorithm should take into account in case of next execution.
 - See (TS3) for the format description.
 - The `subjectColumnPositions` are a map from knowledge bases to column positions.
- Query (7) returns the feedback set.
- Query (8) returns the structured input of the task, upon which the feedback can be set.
 - See (TS4) for the format description.
- Query (9) submits the task for execution.
 - Any previous executions must be either finished or cancelled.
 - The execution object format is described in (TS5), the draft flag is ignored for now. Returns 409 Conflict when the task is already running.
- Query (10) cancels task execution. Returns 409 Conflict when the task has already finished.
- Query (11) returns the result of task execution.
 - The result contains the annotations in structured format as described in (TS6).
 - The call waits for the process to finish.
 - There is also an option to PUT the modified result back to server. This usage is discouraged, but some clients may use it to store work in progress when they do

not have a capability to keep the results themselves.

- Query (12) exports the result (when available) in a chosen (by the MIME type in the Accept header of the request - *text/turtle* and *application/ld+json* are supported) format.
- Query (13) return the state of execution.
 - See (TS7) for format description. Notice the reduction and change of name of some states.
- Query (14) returns a list of all tasks.
 - When the state query parameter is present and set to true, than the tasks aggregated with their states are provided: see (TS8) for format description.
 - The state parameter can be left out or set to false, then the results is just an array of individual tasks as described in (TS1).
 - When the `orderBy` parameter is present and set to `id` or if it is missing, the tasks are sorted by their IDs in ascending order. This parameter can be also set to `created`, then the tasks are sorted by modification time in descending order.

(TS1)

```
{ "id": "task_id", created: "yyyy-MM-dd HH:mm", "configuration":  
  { ... }, "description": "a simple testing task" }
```

For configuration object description, see below.

(TS2)

```
{ "input": "file description ID", feedback: { ... }, "usedBases": [  
  { "name": "DBpedia" }, { "name": "DBpedia German" } ],  
  "primaryBase": { "name": "DBpedia" }, "rowsLimit": 345,  
  "statistical" : false }
```

(TS3) (All indices are zero-based)

```
{  
  "subjectColumnsPositions": { "DBpedia": [ { position: { index:  
  5 } }, ... ], "wikidata": [ { position: { index:  
  2 } }, ... ], ...},  
  "columnIgnores": [ // Columns are ignored for the rest of  
  processing.
```

```

        { position: { index: 6 } },
        ...
    ],
    "columnCompulsory": [ // Columns are compulsorily processed
(class./disamb.) as named entities (even when they are non-named
entities).
        { position: { index: 5 } },
        ...
    ],
    "classifications": [
        {
            position: { index: 5 },
            annotation: {
                "candidates": { // Should contain initial and even
the proposed candidates, even if they were not chosen.
                    "dbpedia": [
                        {
                            "entity": { "resource":
"http://example.com/hoho/Lala", "label": "Ble" },
                            "score": { "value": 0.5 },
                        },
                        ...
                    ],
                    "wikidata": ...,
                    ...
                },
                "chosen": {
                    "dbpedia": [
                        {
                            "entity": { "resource":
"http://example.com", label: "Ble" },
                            "score": { "value": 0.5 }
                        },
                        ...
                    ],
                    "wikidata": ...,
                    ...
                }
            }
        },
        ...
    ],
    "columnAmbiguities": [ // Skipping disambiguation in all cells
of the specified columns.
        { position: { index: 6 } },
        ...
    ],
    "ambiguities": [ // Skipping disambiguation for the cells
provided.

```

```

        { position: { rowPosition: { index: 6 }, columnPosition:
{ index: 6 } } },
        ...
    ],
    "disambiguations": [
        {
            position: { rowPosition: { index: 5 }, columnPosition:
{ index: 9 } },
            annotation: { ... } // Same as above.
        },
        ...
    ],
    "columnRelations": [
        {
            position: { first: { index: 5 }, second: { index:
9 } },
            annotation: { ... } // Same as above.
        },
        ...
    ],
    "dataCubeComponents": [
        {
            position: { index: 5 },
            annotation: {
                "component": {
                    "dbpedia": "DIMENSION", // Possible values:
"DIMENSION" or "MEASURE" or "NONE".
                    "wikidata": ...,
                    ...
                },
                "predicate": {
                    "dbpedia": [
                        {
                            "entity": { "resource":
"http://example.com", label: "Ble" },
                            "score": { "value": 1.0 }
                        },
                        ...
                    ],
                    "wikidata": ...,
                    ...
                }
            }
        },
        ...
    ]
}

```

(TS4)

```
{ "headers": [ "country", "city", ... ], "rows": [ [ "Albania",
"Tirana" ], ... ] }
```

(TS5)

```
{ draft: false }
```

(TS6)

Suppose the following input table that will be processed by the algorithm:

City	Country	Population	Rating
Paris	France	2,240,621	7.9
Prague		1,267,449	8.1
Melbourne		4,529,500	3.4

The result could look like this:

Code Block 1 Result format

```
{
  // Subject columns
  "subjectColumnsPositions" : { "wikidata": [ { position:
{ index: 2 } }], ... ], "yago": [ { position: { index:
5 } }], ... ] },

  "warnings": [ "Failed to disambiguate 7893739ye9y7yad", "Beware
of JAG-JAG bird!" ]

  // Column headers annotations
  "headerAnnotations" : [
    // City
    {
      "candidates" : {
        "wikidata" : [
          {
            "entity": {
              "resource" :
"http://www.wikidata.org/wiki/Q515",
              "label" : "city" ,
              "prefixed" : "wikidata:Q515",
              "prefix" : { "with" :
"wikidata", "what" : "http://www.wikidata.org/wiki/" },
```

```

        "tail" : "Q515"
      },
      "score" : 0.79
    }
  ],
  "yago" : [
    {
      "entity" : {
        "resource" :
"http://yago.org/city",
        "label" : "city"
      },
      "prefixed" : "yago:city",
      "prefix" : { "with" : "yago",
"what" : "http://yago.org/" },
      "tail" : "city"
    },
    "score" : 0.5
  ]
},
"chosen" : {
  "wikidata" : [],
  "yago" : [
    {
      "entity" : {
        "resource" :
"http://yago.org/city",
        "label" : "city"
      },
      "prefixed" : "yago:city"
      "prefix" : { "with" : "yago",
"what" : "http://yago.org/" },
      "tail" : "city"
    },
    "score" : 0.5
  ]
}
},
// Country
{
  "candidates" : {
    "wikidata" : [
      {
        "entity" : {
          "resource" :
"http://www.wikidata.org/wiki/Q6256",
          "label" : "country",
          "prefixed" : "wikidata:Q6256"

```

```

        "prefix" : { "with" :
"wikidata", "what" : "http://www.wikidata.org/wiki/" },
        "tail" : "Q6256"
    },
    "score" : 0.91
},
{
    "entity" : {
        "resource" :
"https://www.wikidata.org/wiki/Q7275",
        "label" : "state",
        "prefixed" : "wikidata:Q7275",
        "prefix" : { "with" :
"wikidata", "what" : "http://www.wikidata.org/wiki/" },
        "tail" : "Q7275"
    },
    "score" : 0.65
}
],
"yago" : [
    {
        "entity" : {
            "resource" :
"http://www.yago.org/country",
            "label" : "country",
            "prefixed" : "yago:country",
            "prefix" : { "with" : "yago",
"what" : "http://www.yago.org/" },
            "tail" : "country"
        },
        "score" : 0.75
    }
]
},
"chosen": { "wikidata" : [], "yago": [] }
},
// Population
{
    "candidates" : {
        "wikidata" : [
            {
                "entity" : {
                    "resource" :
"http://www.wikidata.org/wiki/Q33829",
                    "label" : "population"
                    "prefixed" : "wikidata:Q33829",
                    "prefix" : { "with" :
"wikidata", "what" : "http://www.wikidata.org/wiki/" },

```



```

        "tail" : "Q33829"
      },
      "score" : 0.88
    }
  ]
},
"chosen": { "wikidata" : [], "yago": [] }
},
// Rating
{
  "candidates" : {},
  "chosen": { "wikidata" : [], "yago": [] }
}
],
// Cells annotations
"cellAnnotations" : [
  // 1st row
  [
    // Paris
    {
      "candidates" : {
        "wikidata" : [
          {
            "entity" : {
              "resource" :
"https://www.wikidata.org/wiki/Q90",
              "label" : "Paris",
              "prefixed" :
"wikidata:Q90",
              "prefix" : { "with" :
"wikidata", "what" : "http://www.wikidata.org/wiki/" },
              "tail" : "Q90"
            },
            "score" : 0.92
          }
        ]
      },
      "chosen": { "wikidata" : [] }
    },
    // France
    {
      "candidates" : {
        "wikidata" : [
          {
            "entity" : {
              "resource" :

```

```

"http://www.wikidata.org/wiki/Q142",
    "label" : "France",
    "prefixed" :
"wikidata:Q142",
    "prefix" : { "with" :
"wikidata", "what" : "http://www.wikidata.org/wiki/" },
    "tail" : "Q142"
    },
    "score" : 0.92
    }
    ]
    },
    "chosen": { "wikidata" : [] }
},
// 2,240,621
{
    "candidates" : { "wikidata" : [] },
    "chosen": { "wikidata" : [] }
},
// 7.9
{
    "candidates" : {"wikidata" : []},
    "chosen": { "wikidata" : [] }
}
],
// 2nd row
[
    // Prague
    {
        "candidates" : {
            "wikidata" : [
                {
                    "entity" : {
                        "resource" :
"http://www.wikidata.org/wiki/Q1085",
                        "label" : "Prague",
                        "prefixed" :
"wikidata:Q1085",
                        "prefix" : { "with" :
"wikidata", "what" : "http://www.wikidata.org/wiki/" },
                        "tail" : "Q1085"
                    },
                    "score" : 0.92
                }
            ]
        }
    }
]
}

```

```

    },
    // Empty
    {
        "candidates" : {"wikidata" : []}
    },
    // 1,267,449
    {
        "candidates" : {"wikidata" : []}
    },
    // 8.1
    {
        "candidates" : {"wikidata" : []}
    }
],
// 3rd row
[
    // Melbourne
    {
        "candidates" : {
            "wikidata" : [
                {
                    "entity" : {
                        "resource" :
"https://www.wikidata.org/wiki/Q3141",
                        "label" : "Melbourne",
                        "prefixed" :
"wikidata:Q3141",
                        "prefix" : { "with" :
"wikidata", "what" : "http://www.wikidata.org/wiki/" },
                        "tail" : "Q3141"
                    },
                    "score" : 0.92
                }
            ]
        }
    },
    // Empty
    {
        "candidates" : {}
    },
    // 4,529,500
    {
        "candidates" : {}
    }
]

```

```

        },
        // 3.4
        {
            "candidates" : {}
        }
    ],
],
// Relations between columns (may be sparse)
"columnRelationAnnotations": {
    // Indicates relations for the 1st column.
    "0" : {
        // Indicates relations between the 1st and 3rd column
        "2" : {
            "candidates" : {
                "wikidata" : [
                    {
                        "entity" : {
                            "resource" :
"http://www.wikidata.org/wiki/Property/P1082",
                            "label" : "has
population",
                            "prefixed" :
"http://www.wikidata.org/wiki/Property/P1082",
                            "prefix" : { "with" :
"http://www.wikidata.org/wiki/Property/" },
                            "tail" : "P1082"
                        },
                        "score" : 0.9
                    }
                ]
            },
            "chosen" : {
                "wikidata" : [
                    {
                        "entity" : {
                            "resource" :
"http://www.wikidata.org/wiki/Property/P1082",
                            "label" : "has
population",
                            "prefixed" :
"http://www.wikidata.org/wiki/Property/P1082",
                            "prefix" : { "with" :
"http://www.wikidata.org/wiki/Property/" },
                            "tail" : "P1082"
                        }
                    }
                ]
            }
        }
    }
}

```

```

        },
        "score" : 0.9
    }
    ]
}
},
// Statistical annotations
"statisticalAnnotations" : [
    // City
    {
        "component" : {
            "wikidata" : "DIMENSION",
            "yago" : "DIMENSION"
        },
        "predicate" : {
            "wikidata" : [],
            "yago" : []
        }
    },
    // Country
    {
        "component" : {
            "wikidata" : "DIMENSION",
            "yago" : "DIMENSION"
        },
        "predicate" : {
            "wikidata" : [
                {
                    "entity" : {
                        "resource" :
"http://www.wikidata.org/wiki/Q6256",
                        "label" : "country",
                        "prefixed" : "wikidata:Q6256",
                        "prefix" : { "with" :
"wikidata", "what" : "http://www.wikidata.org/wiki/" },
                        "tail" : "Q6256"
                    },
                    "score" : 1.0
                }
            ],
            "yago" : []
        }
    },
}
},

```

```

    // Population
    {
        "component" : {
            "wikidata" : "MEASURE",
            "yago" : "MEASURE"
        },
        "predicate" : {
            "wikidata" : [
                {
                    "entity" : {
                        "resource" :
"http://www.wikidata.org/wiki/Q33829",
                        "label" : "population",
                        "prefixed" : "wikidata:Q33829",
                        "prefix" : { "with" :
"wikidata", "what" : "http://www.wikidata.org/wiki/" },
                        "tail" : "Q33829"
                    },
                    "score" : 1.0
                }
            ],
            "yago" : []
        }
    },

    // Rating
    {
        "component" : {
            "wikidata" : "NONE",
            "yago" : "NONE"
        },
        "predicate": { "wikidata" : [], "yago": [] }
    }
],

// Column processing annotations
"columnProcessingAnnotations" : [
    // City
    {
        "processingType" : {
            "wikidata" : "COMPULSORY",
            "yago" : "COMPULSORY"
        }
    },

    // Country
    {
        "processingType" : {
            "wikidata" : "NAMED_ENTITY",

```

```

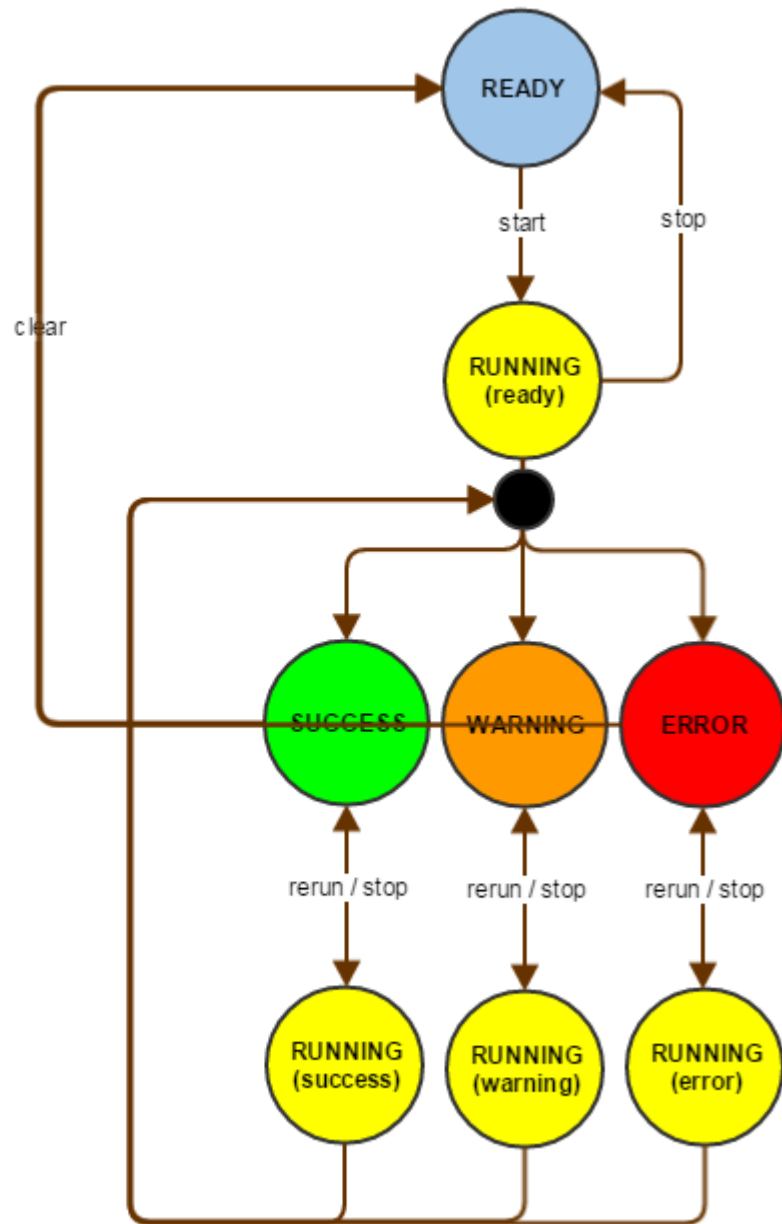
        "yago" : "NAMED_ENTITY"
    },
    // Population
    {
        "processingType" : {
            "wikidata" : "NON_NAMED_ENTITY",
            "yago" : "NON_NAMED_ENTITY"
        }
    },
    // Rating
    {
        "processingType" : {
            "wikidata" : "IGNORED",
            "yago" : "IGNORED"
        }
    }
]
}

```

(TS7)

READY, RUNNING, SUCCESS, WARNING, ERROR

READY stands for created, not run yet, or cancelled task. RUNNING is self-explanatory, SUCCESS means that result is ready, WARNING indicates that the result is ready with warnings, but requires further user's attention. Finally ERROR means no result was generated; the algorithm halted halfway with fatal error.



(TS8)

Code Block 2 TS8

```

[
  {
    "id": "task_name1",
    "description": "a simple testing task",
  }
]

```



```

    "created": "2016-01-01 09:58",

    // Status of all tasks should be present. It is expected UI
    // will be polling only those that are in progress
    "status": "READY", // Follows the state convention.
    "configuration": {
        ...
    },
    {
        /* ... */
    }
]

```

4.1.7 Bases

(1) **GET** <http://odalic.eu/odalic/bases?modifiable={true|false}>

- - Returns array of available configured knowledge bases, see **(B1)** for payload example (and **(B2)** for a detail of a single element).
 - The valid values for textSearchingMethod are: fulltext, substring, exact.
 - When **modifiable** is set to true (default, when not set, is false), only knowledge bases supporting proposal (via insertion) are listed.

(2) **GET** <http://odalic.eu/odalic/bases/{name}>

- - Returns a specific knowledge base configuration, see **(B2)** for payload example.
 - When the accepted MIME type is text/turtle instead of application/json, it exports the knowledge base in RDF serialized in to Turtle.

(3) **PUT** <http://odalic.eu/odalic/bases/{name}>

- - Sets the specific knowledge base configuration, see **(B2)** for payload example.

- When the content MIME type is text/turtle instead of application/json, the servers assumes import of the knowledge base in the RDF format (mechanism similar to Tasks import).

(4) **DELETE** <http://odalic.eu/odalic/bases/{name}>

-

- Removes the specific knowledge base configuration.
- May fail with 409 Conflict when in use by some task.

(B1)

```
[ {"name" : "German DBpedia", ... }, ... ]
```

(B2)

```
{ "name" : "German DBpedia", "endpoint" :
  "http://de.dbpedia.org/sparql", "description" : "German version of
  DBpedia", "textSearchingMethod" : "fulltext", "languageTag" : "en",
  "skippedAttributes" : [ "http://www.w3.org/ns/prov#wasDerivedFrom",
  ... ], "skippedClasses" :
  [ "http://www.w3.org/2002/07/owl#Thing", ... ],
  "groupsAutoSelected" : false, "selectedGroups" : [ "RDF", ... ],
  "insertEnabled" : true, "insertEndpoint" :
  "http://de.dbpedia.org/sparql/insert", "insertGraph" :
  "http://odalic.eu", "userClassesPrefix" : "schema",
  "userResourcesPrefix" : "resource", "objectProperty" :
  "owl:ObjectProperty", "datatypeProperty" : "owl:DatatypeProperty",
  "login" : "bob", "password" : "1234", "advancedType" : "SPARQL",
  "advancedProperties" : { "eu.odalic.custom.key" : "custom
  value", ... } }
```

4.1.8 Advanced base types

(1) **GET** <http://odalic.eu/odalic/advanced-base-types>

-

- Returns array of available advanced base types, see **(ABT1)** for a detail of a single element.

- Shared by all users, accessible without users/{userId} part.

(2) **GET** <http://odalic.eu/odalic/advanced-base-types/{name}>

-

- Returns a specific knowledge base configuration, see **(ABT1)** for payload example.
- Shared by all users, accessible without users/{userId} part.

(ABT1)

```
{ "name" : "SPARQL", "keys" : [ "eu.odalic.custom.key",  
  "eu.odalic.another.key", ... ], "keysToDefaultValues" :  
  { "eu.odalic.another.key" : "Default value", ... },  
  "keysToComments" : { "eu.odalic.custom.key" : "Comment  
  example..." } }
```

4.1.9 Predicates and classes groups

(1) **GET** <http://odalic.eu/odalic/groups>

-

- Returns array of user's defined groups, see **(G1)** for a detail of a single element.

-

- Returns a specific user's defined group, see **(G1)** for payload example.

(2) **GET** <http://odalic.eu/odalic/groups/{groupId}>

(3) **PUT** <http://odalic.eu/odalic/groups/{groupId}>

-

- Sets user's defined group.
- See **(G1)** for payload example.

(4) **DELETE** <http://odalic.eu/odalic/groups/{groupId}>

-

- Removes the user-defined group.
- May fail when the group is used in some base definition with 409 Conflict.

(G1)

```
{ "id" : "DBpedia", "labelPredicates" :  
  [ "http://dbpedia.org/property/name", ... ],  
  "descriptionPredicates" :  
  [ "http://dbpedia.org/ontology/abstract" ],  
  "instanceOfPredicates" : [ "http://www.w3.org/1999/02/22-rdf-  
syntax-ns#type" ], "classTypes" : [ "http://www.w3.org/2000/01/rdf-  
schema#Class", ... ], "propertyTypes" :  
  [ "http://www.w3.org/1999/02/22-rdf-syntax-ns#Property", ... ] }
```

4.1.10 Entities

- (1) **GET** `http://odalic.eu/odalic/bases/{base}/entities?query={needle}&limit={20}`
 - Returns array of entities (not discriminating classes, resources, properties,...) in the same format as in the Result (without the scores naturally), see (ES1).
 - The **base** is the knowledge base where the search takes place.
 - The **query** is a URL parameter of the string searched for.
 - The **limit** is the maximum count of provided hits. Can be omitted and then 20 is used as default.
 - **Deprecated.**
- (2) **GET** `http://odalic.eu/odalic/bases/{base}/entities/classes?query={needle}&limit={20}`
 - Returns array of classes in the same format as in the Result (without the score naturally), see (ES1).
 - The **prefixed** field contains the same content as the resource field, if the prefix is not defined on the server for the resource URI. Otherwise the **prefixed** contains the resource URI shortened by the use of prefix defined at `config/PrefixMapping.ttl`.
 - The prefix is always separated by a colon.
 - When sent to server, the prefixed field is optional and ignored by the server.
 - The **prefix** field contains an object consisting of **with** (the actual prefix string) and **what** (associated URI) attributes. It is also optional and ignored when sent to server.
 - If the **prefix** is not defined, it is null or not even present.

- The **tail** field contains the rest of the string after the prefix separator, when the prefix is defined. Otherwise it is null. It is also optional and ignored when sent to server.
 - The **base** is the knowledge base where the search takes place.
 - The **query** is a URL parameter of the string searched for.
 - The **limit** is the maximum count of provided hits. Can be omitted and then 20 is used as default.
- (3) **GET** `http://odalic.eu/odalic/bases/{base}/entities/resources?query={needle}&limit={20}`
 - Returns array of resources in the same format as in the Result (without the score naturally), see (ES1).
 - The **base** is the knowledge base where the search takes place.
 - The **query** is a URL parameter of the string searched for.
 - The **limit** is the maximum count of provided hits. Can be omitted and then 20 is used as default.
- (4) **GET** `http://odalic.eu/odalic/bases/{base}/entities/properties?query={needle}&limit={20}`
 - Returns array of properties in the same format as in the Result (without the score naturally), see (ES1).
 - The **base** is the knowledge base where the search takes place.
 - The **query** is a URL parameter of the string searched for.
 - The **limit** is the maximum count of provided hits. Can be omitted and then 20 is used as default.

(ES1)

```
[ {"resource" : "http://www.wikidata.org/wiki/Property/P1082",
  "label" : "has population", "prefixed" : "wikidataprop:P1082",
  "prefix" : { "with" : "wikidatapro", "what" :
  "http://www.wikidata.org/wiki/Property/" }, "tail":
  "P1082" }, ... ]
```

4.1.11 Proposing entities

- (1) **POST** `http://odalic.eu/odalic/bases/{base}/entities/classes`
 - For payload format see (EP1).
 - The `base` is the knowledge base where the proposed class will be created. It must support insertion.
 - Returns a newly created class, if nothing bad happens. Format conforms to (ES1).
- (2) **POST** `http://odalic.eu/odalic/bases/{base}/entities/resources`
 - For payload format see (EP2).
 - The `base` is the knowledge base where the proposed resource will be created. It must support insertion.
 - Returns a newly created entity, if nothing bad happens. Format conforms to (ES2).
- (3) **POST** `http://odalic.eu/odalic/bases/{base}/entities/properties`
 - For payload format see (EP3).
 - The `base` is the knowledge base where the proposed property will be created. It must support insertion.
 - Returns a newly created entity, if nothing bad happens. Format conforms to (ES3) with the exception that `domain` and `range` are currently ignored.
 - The type attribute value is either "data" (indicating "Data-type property") or "object" (for "Object-type" property), with "Object-type" being the default option when the type is missing.

(EP1)

```
{ "label": "Country", "alternativeLabels": ["State", ...], suffix:
  "Class:C1033", "superClass" : { "resource" :
  "https://www.wikidata.org/wiki/Class:C999", "label" : "State
  formation" } }
```

(EP2)

```
{ "label": "Prague(city)", "alternativeLabels": ["The capital city
```

```
of Prague"], suffix: "Resource:R82123", "classes" : [{"resource" :  
"https://www.wikidata.org/wiki/Class:C1080", "label" :  
"City" }, ...]}
```

(EP3)

```
{"label": "capital", "alternativeLabels": ["seat of government"],  
suffix: "Property:P23778", "superProperty" : {"resource" :  
"https://www.wikidata.org/wiki/Property:P99", "label" : "political  
centre" }, "domain" : {"resource" :  
"https://www.wikidata.org/wiki/Class:C1080", "label" : "City" },  
"range" : {"resource" : "https://www.wikidata.org/wiki/Class:C999",  
"label" : "State formation" }, "type" : "data"}
```

5 UnifiedViews DPU

5.1.1 Introduction

The UnifiedViews DPU allows the user to run and schedule [Odalic Semantic Table Interpretation](#) in combination with other plugins in [UnifiedViews](#). In order to get yourself familiarized with UnifiedViews and its concepts and usage follow the [documentation](#).

5.1.2 Setting up a template

5.1.2.1 Retrieving authentication token

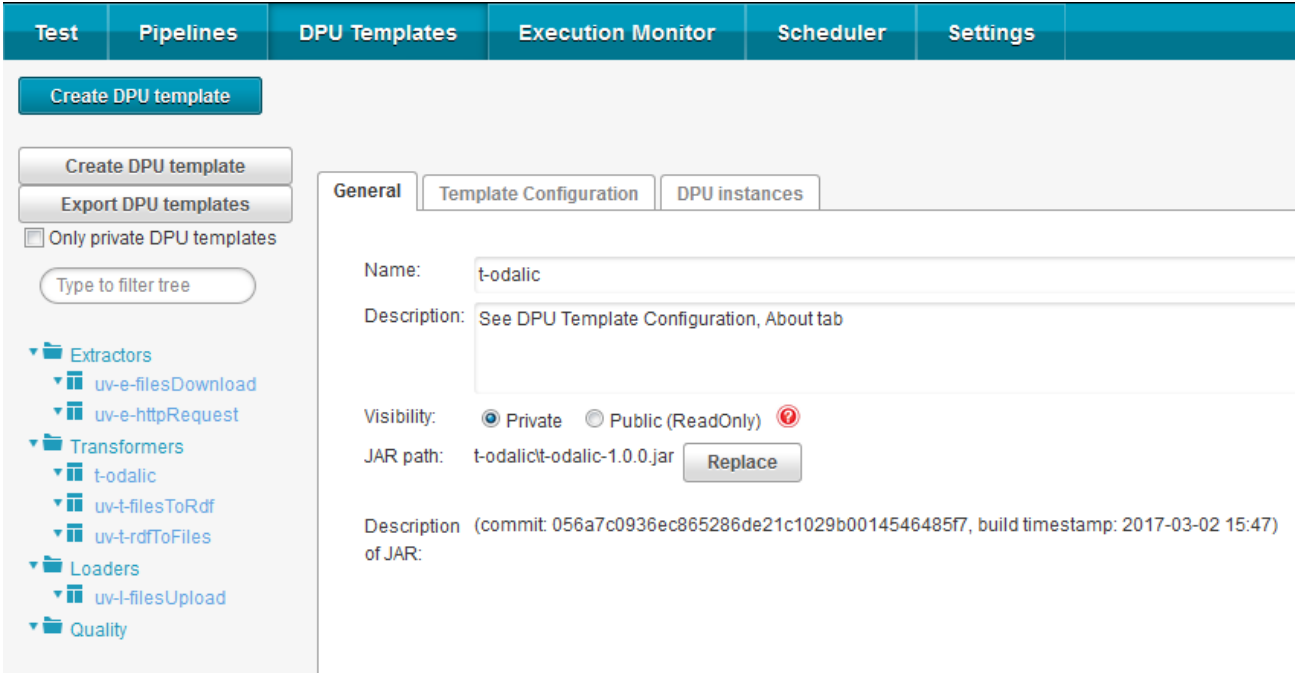
First obtain the authentication token. This can be done by accessing a web client that will share the same server as the plugin. Log in and go to **My account** and click **Show token** to reveal it. Copy the token to clipboard.

👤 Logged in as odalic@email.cz. Log out? Change password? [Show token?](#)

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpzZW50L3N1bWVhcnR5IiwiaWF0Ijoi
1480000000.eyJpc3MiOiJZGFsaWVhcnR5IiwiaWF0IjoiE0ODkwOTAwNzks
Imp0aSI6ImY3Y2MyNGFkLTQzNGMtNGU2Zi1iZmU4LWVhcnR5IiwiaWF0Ijoi
E2NSJ9.4Z4MO0A8YDu0E9dQPqQ6Dcgr5tln3D8EDdCaz_20iW
```

5.1.2.2 Accessing plugin properties

Following the [plugin installation](#) , go to DPU Templates if you are not already there, and open the **t-odalic** plugin properties by selecting it in the left column.



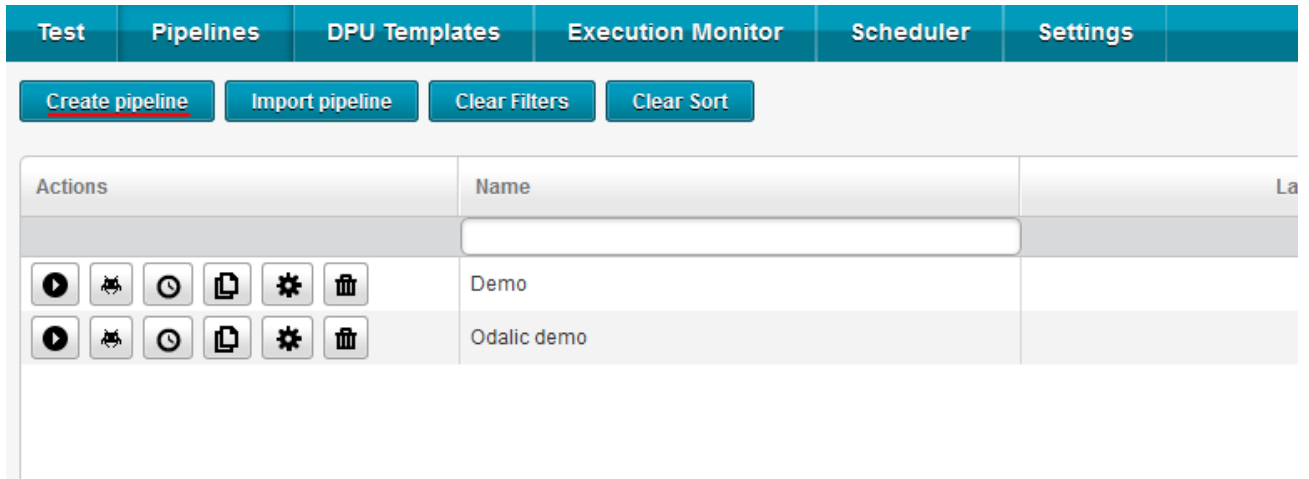
5.1.2.3 Template configuration

Go to the **Template configuration** tab and insert the Odalic server address to **Host** and paste the obtained token to the **Authentication token** field. You can also set the other settings, but it is usually better to set them for each instance individually.

5.1.3 Using the instances

5.1.3.1 Creating pipeline

To use the set-up DPU instance you have to create a pipeline first. Go to **Pipelines** and click **Create pipeline**.

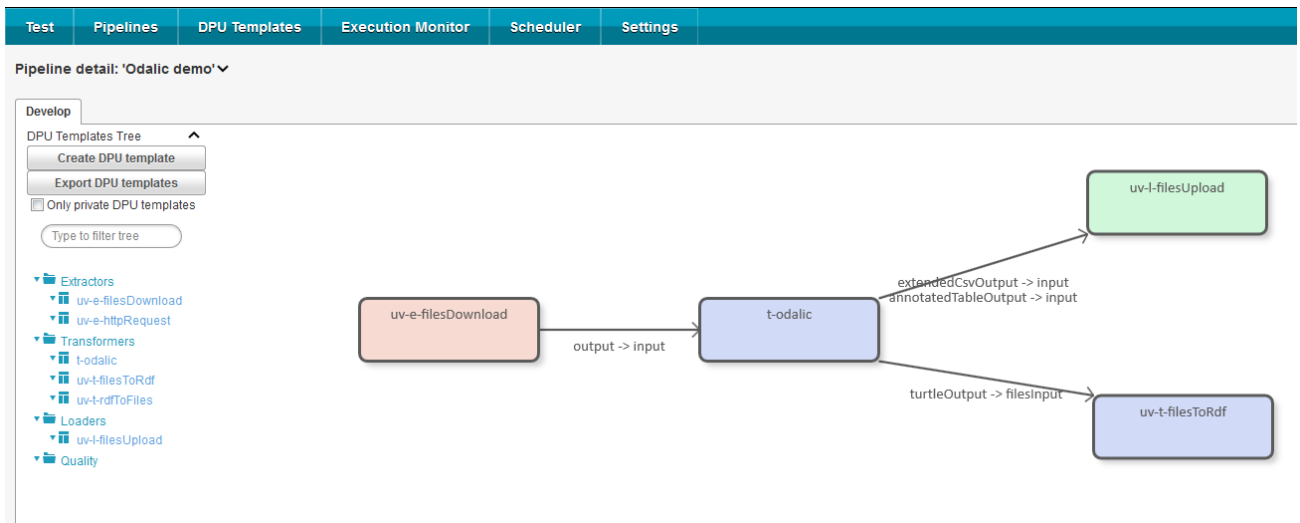


5.1.3.2 Pipeline workbench

A workbench to construct UnifiedViews pipeline appears, with the list of available DPU templates on the left. Pick the t-odalic template and drag it to the working plane and connect it with other desired DPUs.

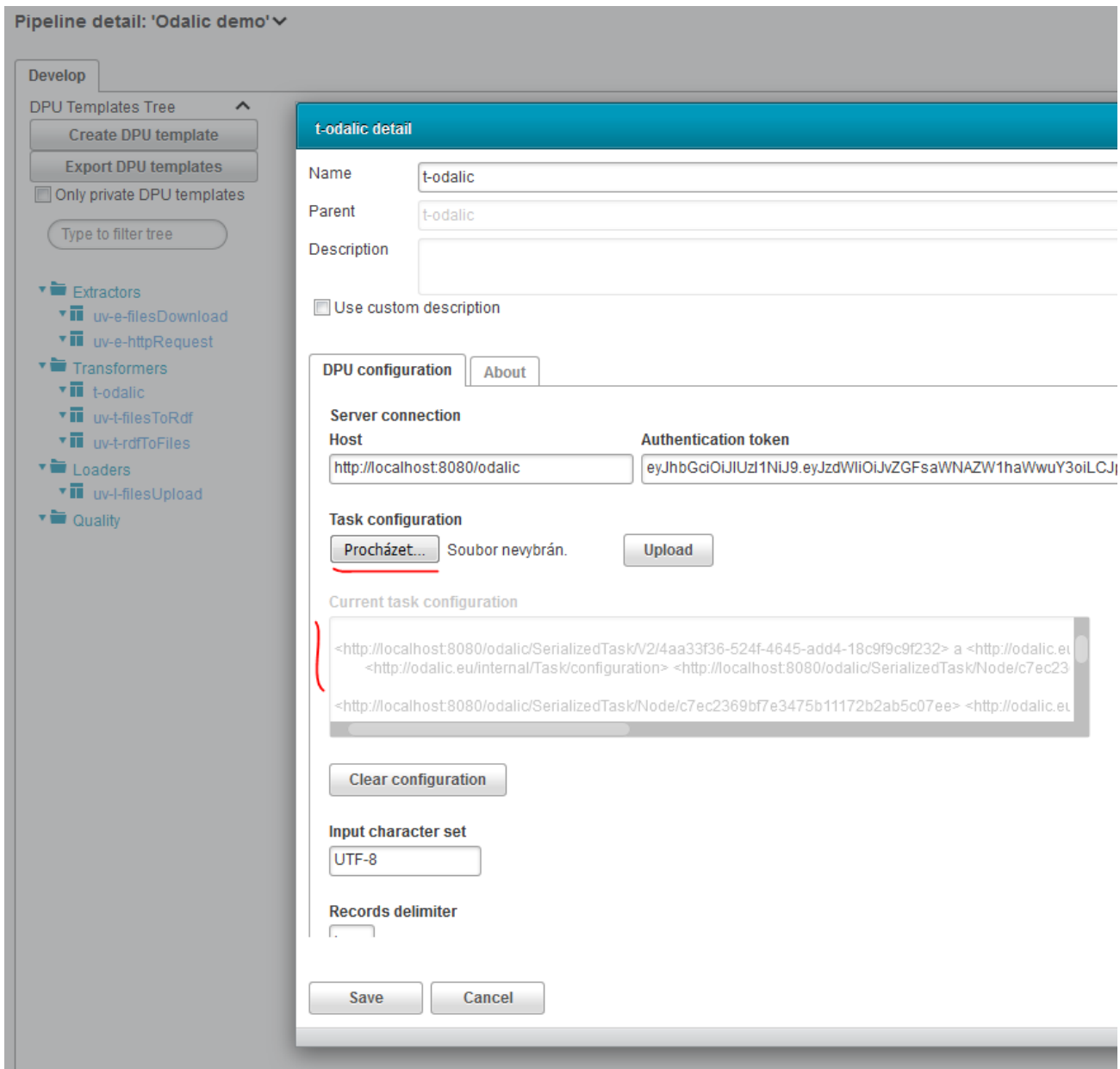
The Odalic DPU accepts only one mandatory file input and produces three file outputs: *extended CSV* and the complementary *table annotations* and *serialized RDF in Turtle format*. All three outputs are optional, that means they do not have to be connected to an input of another DPU.

The following figure demonstrates the use of plugin by connecting its input to a DPU downloading file from a remote location and its outputs to two plugins: the first one will upload the extended CSV and the annotated table to some location, the second one will parse the Turtle input into RDF model, which the user can for example connect to some RDF storage or query with SPARQL DPUs.



5.1.3.3 Instance configuration

Now is the time to set the instance of the t-odalic by clicking the gear icon which appears when hovering with a mouse over the associated blue box. A dialog similar to the one configuring the template appears. [Export a task configuration](#) from the web client to make it serve as the blueprint for this DPU processing. Upload it using the **Task configuration** form component. Its content will appear in the non-editable text area. You can **Clear** the **configuration** and upload another if necessary. **Save** the instance specific settings.



5.1.3.4 Debugging the pipeline

You can debug the sample pipeline fragment by hitting the *bug icon* on the *uv-t-filesToRdf* DPU box. A reporting window will appear and after the processing finishes, it should look something like the following figure. Notice the lines that have *t-odalic* in the DPU Instance column, these are the messages sent by the Odalic DPU about its processing state.

Debug window

Pipeline status: ✔

Events | Log | Browse/Query

type	Timestamp	DPU Instance	Short message
i	Mar 2, 2017 5:16:18 F		Starting execution: 29
i	Mar 2, 2017 5:16:18 F	uv-e-filesDownload	Starting DPU developer's code for DPU: 20
i	Mar 2, 2017 5:16:19 F	uv-e-filesDownload	Correctly processed input entries : 1 / 1 .
i	Mar 2, 2017 5:16:19 F	uv-e-filesDownload	Correctly downloaded files : 1 / 1
i	Mar 2, 2017 5:16:19 F	uv-e-filesDownload	DPU: 20 completed.
i	Mar 2, 2017 5:16:19 F	t-odalic	Starting DPU developer's code for DPU: 19
i	Mar 2, 2017 5:16:19 F	t-odalic	Odalic DPU is running ...
i	Mar 2, 2017 5:16:19 F	t-odalic	Sending input...
i	Mar 2, 2017 5:16:20 F	t-odalic	Input sent.
i	Mar 2, 2017 5:16:20 F	t-odalic	Sending task configuration...
i	Mar 2, 2017 5:16:20 F	t-odalic	Task configuration sent.
i	Mar 2, 2017 5:16:20 F	t-odalic	Starting execution...
i	Mar 2, 2017 5:16:29 F	t-odalic	Success!
i	Mar 2, 2017 5:16:29 F	t-odalic	Execution finished.
i	Mar 2, 2017 5:16:29 F	t-odalic	Exporting to outputs...
i	Mar 2, 2017 5:16:29 F	t-odalic	Exporting finished.
i	Mar 2, 2017 5:16:29 F	t-odalic	Odalic DPU processing finished.
i	Mar 2, 2017 5:16:29 F	t-odalic	DPU: 19 completed.
i	Mar 2, 2017 5:16:30 F	uv-t-filesToRdf	Starting DPU developer's code for DPU: 21
i	Mar 2, 2017 5:16:30 F	uv-t-filesToRdf	Addon failed to load configuration

Number of Records:23

Page: 1 / 2

Rerun | Close

5.1.3.5 Result preview

You can preview the parsed RDF output which was produced during the semantic table interpretation by going to **Browse/Query** tab, selecting the *uv-t-filesToRdf* DPU and choosing its *rdfOutput*. A table of parsed triples like on the following figure should appear.

The screenshot shows the 'Debug window' interface. At the top, it indicates 'Pipeline status: ✓'. Below this, there are tabs for 'Events', 'Log', and 'Browse/Query'. The 'Browse/Query' tab is active, showing a 'Select DPU' dropdown set to 'uv-t-filesToRdf' and a 'Select Data Unit' dropdown set to 'rdfOutput'. A 'Browse' button is next to the 'rdfOutput' dropdown. Below this, the URL 'http://unifiedviews.eu/resource/internal/dataunit/exec/29/dpu/21/du/1' is displayed. A 'SPARQL Query:' section contains a text area with the query: 'CONSTRUCT {?s ?p ?o} WHERE {?s ?p ?o} LIMIT 1000'. To the right of the query area are buttons for 'Run Query', a format dropdown set to 'TTL', and 'Run Query and Download'. Below the query area is a table with three columns: 'subject', 'predicate', and 'object'. The table contains 14 rows of RDF triples. At the bottom of the window, there is a 'Page: 1 / 3' indicator and a 'Number of Records:41' label. A 'Download' button is also visible at the bottom left.

subject	predicate	object
http://dbpedia.org/resource/Deadhouse_Gates	http://dbpedia.org/ontology/author	http://dbpedia.org/resource/Steven_Erikson
http://dbpedia.org/resource/Deadhouse_Gates	http://dbpedia.org/ontology/series	http://dbpedia.org/resource/Malazan_Book_of_the_Fallen
http://dbpedia.org/resource/Deadhouse_Gates	http://purl.org/dc/terms/title	Deadhouse Gates
http://dbpedia.org/resource/Deadhouse_Gates	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://schema.org/Book
http://dbpedia.org/resource/Deadhouse_Gates	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://xmins.com/foaf/0.1/Person
http://dbpedia.org/resource/Deadhouse_Gates	http://www.w3.org/2002/07/owl#sameAs	http://dbpedia.org/resource/Deadhouse_Gates
http://dbpedia.org/resource/Steven_Erikson	http://purl.org/dc/terms/title	Steven Erikson
http://dbpedia.org/resource/Steven_Erikson	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://xmins.com/foaf/0.1/Person
http://dbpedia.org/resource/Steven_Erikson	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://dbpedia.org/class/yago/WikicatCanadianFantasyWriters
http://dbpedia.org/resource/Steven_Erikson	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.ontologydesignpatterns.org/ont/dul/DUL_owl#NaturalPerson
http://dbpedia.org/resource/Steven_Erikson	http://www.w3.org/2002/07/owl#sameAs	http://dbpedia.org/resource/Steven_Erikson
http://dbpedia.org/resource/Steven_Erikson	http://www.w3.org/2002/07/owl#sameAs	http://de.dbpedia.org/resource/Steven_Erikson
http://dbpedia.org/resource/Malazan_Book_of_the_Fallen	http://purl.org/dc/terms/title	Malazan Book of the Fallen
http://dbpedia.org/resource/Malazan_Book_of_the_Fallen	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://schema.org/Book
http://dbpedia.org/resource/Malazan_Book_of_the_Fallen	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://dbpedia.org/ontology/Work

In order to have a valid pipeline you must connect the RDF parsing plugin to some output DPU. Nevertheless even this small example should give you a good grasp on how to tackle the semantic table interpretation with UnifiedViews using the Odalic plugin.

6 Glossary

6.1.1 W3C Linked Data Glossary

<https://www.w3.org/TR/ld-glossary/>

6.1.2 Project glossary

- administrator
 - user with elevated rights (mainly can import knowledge bases, monitor other users' tasks)
- Austrian open data catalogs
 - data portals that serve as the source of the test data
 - <https://www.data.gv.at>
 - additionally <https://www.opendataportal.at/>
- cell disambiguation
 - determining which RDF resource is represented by a literal string in the cell
- columns classification
 - column classification annotates each NE-column with one concept, or in the case of literal columns, associates the column to one property of the concept assigned to the subject column of the table
- conversion
 - the same as 'task'
- CSV meta-data
 - CSV file description
 - conforming to JSON schema specified at https://www.w3.org/2013/csvw/wiki/Main_Page (see metadata)
 - example here: <http://data.opendataportal.at/dataset/kunstler-der-sammlung-des-mumok/resource/e25640f8-a3e4-46d2-8a4f-9be471b115d2>
- CSV schema
 - see CSV meta-data

- DBpedia.org
 - one of the existing Linked Data knowledge bases
- disambiguated entity
 - result of disambiguation
- disambiguation
 - see cell disambiguation
- execution
 - processing of the input file according with Odalic algorithm based on the TableMiner+
- execution service
 - module providing execution
- feedback
 - user input manually setting some of the result annotations or giving constraints to the algorithm in order to improve results in the next run
- focused knowledge bases
 - knowledge bases containing specialized, usually detailed information about certain, well defined domain
 - opposite of general knowledge bases
- Freebase
 - on of the existing knowledge bases
 - deprecated
- general knowledge bases
 - knowledge bases containing general information without restriction to particular topic
 - opposite of focused knowledge bases
- input constraints
 - see feedback
- JSON+LD
 - JSON based format for encoding linked data
- knowledge base's SPARQL endpoint
 - web service allowing querying the knowledge base using SPARQL

- knowledge bases
 - published and accessible collection of datasets composed of RDF triples (or quads if named graphs are involved) and the accompanying infrastructure allowing to query it, modify, search
- Linked Open Data Cloud
 - <http://lod-cloud.net/>
- literal column
 - column with data literal, e.g., plain string, number
- named graph
 - extension of RDF
 - created by extending triples to quads with additional information described by URI
 - can be used to make the manipulation with RDF sets more flexible (adapted by SPARQL)
 - <https://blog.ldodds.com/2009/11/05/managing-rdf-using-named-graphs/>
- NE-column
 - column with entity (currently, as long as it is a string with letters, it is considered as named entity column)
- ontology
 - a model for describing the world
 - consists of a set of types, properties, and relationship types
 - description of taxonomy, classification network
- OWL
 - Web Ontology Language
 - a family of languages for authoring of ontologies
 - <http://www.cambridgesemantics.com/semantic-university/owl-101>
- owl:sameAs
 - built-in OWL property that links an individual to an individual
 - an owl:sameAs statement indicates that two URI references actually refer to the same thing
- predicate

- second part of an RDF statement
- defines the property for the subject of the statement
- always a URI
- establishes the relationship between a subject and an object and makes the object value a characteristic of the subject.
- visually connects subject and object in an RDF graph
- primary KB
 - KB which is denoted as being the primary one - during export the concepts from such KBs are preferred, and all the possible conflicts are resolved with respect to its heightened priority.
- RDF data model
 - a standard model for data interchange on the WebRDF
 - uses URIs to name the relationship between things as well as the two ends of the link (triple)
 - allows for data merging even if the underlying schemas differ, specifically supports the evolution of schemas over time without requiring all the data consumers to be changed
 - <http://www.cambridgesemantics.com/semantic-university/rdf-101>
- RDF data cube
 - organization of RDF data indexed by its dimensions
 - can be visualized as a hypercube
- RDF data format serialization
 - encoding of RDF triples
 - enables to store them permanently or transport over network
 - many options (Turtle and similar, XML, JSON)
- RDF export configuration
 - Configuration of the way how the classified, disambiguated data with relations discovered, are exported to RDF. The template describing the format of the data exported as RDF must be stored (e.g. in the form of SPARQL triple patterns, which may be applied to all rows as the export is prepared).

-

```
?cell101 rdf:type ad:City; s:address ad:address. ad:address  
s:postalCode ?cell105
```

- RDF store
 - store keeping RDF data
- RDF triple
 - object, predicate and subject
- RDFS
 - RDF Schema
 - semantic extension of RDF
 - provides mechanisms for describing groups of related resources and the relationships between these resources
 - operates with domains and ranges of properties
 - <http://www.cambridgesemantics.com/semantic-university/rdfs-introduction>
- RDFS/OWL
 - <http://www.cambridgesemantics.com/semantic-university/rdfs-vs-owl>
- relation label/predicate
 - predicate assigned to particular binary relation between two NE-columns
- relations discovery and creation
 - identifies binary relations between NE-columns
 - alternatively, in the case of one NE-column and a literal column and given that the NE-column is annotated by a specific concept, identifies a property of that concept that could explain the data literals
- result preview
 - limited (probably to predefined number of processed rows) view of the result computed by the algorithm
 - user may be able to provide feedback based on the preview
 - Odalic allows to limit the number of processed rows in the task configuration
- Semantic Table Interpretation
 - name of the problem that TableMiner+ and consequently Odalic try to solve
- SPARQL

- semantic query language
 - able to retrieve and manipulate data RDF
- SPARQL endpoint
 - a service accepting and returning result of SPARQL queries
- staged file
 - uploaded file (either by direct upload or providing a link to a remote one) destined to be processed by the core algorithm
- subject column
 - assumed to exist in every processed table
 - exactly one per table (unless statistical data processing is selected)
- suggested class/concept
 - the class which was suggested by the algorithm as being one of the possible classes
- suggested winning class/concept
 - the class which was suggested by the algorithm as being the best for the given column
- TableMiner+
 - core files predecessor and the base of extensions
- tabular data
 - abstract data loaded from provided CSV
- task
 - a unit of processing
 - defined by a input content (uploaded file) and task configuration
 - has a defined state and transition between them
 - from the application's point of view, each task is a unique one, even though they can be defined by the same input constraints, configurations and input files
- task configuration
 - set of options allowing to run the task without any further input apart from the actual file
 - includes delimiters, knowledge-bases used, input specification,...
- transformation
 - see task

- Turtle
 - terse encoding format of RDF triples
- UnifiedViews pipeline
 - a set of UV DPU instances connected
- UnifiedViews
 - integration tool for linked data
 - <http://unifiedviews.eu>
 - <https://www.semantic-web.at/unifiedviews>
- UnifiedViews DPU
 - processing unit in the UnifiedViews pipeline
 - provided as a plugin (programmable)
 - examples: download, export, SPARQL query,...
- URI
 - used in context of the Linked Data to identify entities
- user
 - someone who is able to log in to the running Odalic instance
 - can stage files, create and run tasks, stop them, run again
 - can choose from available knowledge bases to apply to particular task
- WikiData
 - one of the existing knowledge bases
 - <https://www.wikidata.org/>
- YAGO
 - one of the existing vocabularies
 - <http://www.mpi-inf.mpg.de/departments/databases-and-information-systems/research/yago-naga/yago/>

7 INSTALLATION GUIDES

Odalic platform consists of three main components: the server application (**Odalic Semantic Table Interpretation**), the web client (**Odalic UI**), and finally the **Odalic UnifiedViews plugin**. Both the web client and the plugin require a running server to work properly.

8 Server installation

8.1.1 Prerequisites

- at least 4 GB RAM
- [Java Runtime Environment](#) version 8+
- [Apache Tomcat](#) servlet container version 8+
- [Apache Maven 3](#) (to build and install from the source files)
- running SMTP server (optimally allowing secure communication) in order to send confirmation e-mails (*optional*)

The server is platform independent and runs on any major operating system supporting Java Runtime Environment.

8.1.2 Deployment

There are basically two ways to install the server application: the user can either use a prepared web archive (**WAR**) from the the [installation disc](#), or compile the server web archive file from the provided source files. In both cases the produced WAR file is to be deployed in a running Apache Tomcat instance.

Apart from the web archive, the server also needs access to a separate working directory `{sti.home}` with subdirectories `config` and `resources`. Its default version can be also obtained from the root of [server source repository](#) (files and directories other than `config` and `resources` are meant to build the project and are not needed in the working directory) or copied directly from the installation disc.

The Java Virtual Machine running the Tomcat itself has to be started with a system property `cz.cuni.mff.xrg.odalic.sti` set to a path leading to the main configuration file, e.g.:

```
-Dcz.cuni.mff.xrg.odalic.sti={sti.home}/config/sti.properties
```

where the `{sti.home}/config/sti.properties` is a location of the main configuration file.

The details of configuration are described in section [Configuration](#), the minimum needed settings are the following (located in the main configuration file `config/sti.properties`):

- `sti.home` - absolute path to the root working directory `{sti.home}`.
- `cz.cuni.mff.xrg.odalic.users.admin.email` - default (admin) user email (used as login)
- `cz.cuni.mff.xrg.odalic.users.admin.password` - default (admin) user password (can be changed later using the API)
- `cz.cuni.mff.xrg.odalic.db.file` - absolute path to a file (which will be created if not existing yet) keeping the state of the server

In the `config/websearch.properties`, set the authorization token at `bing.key` to the Bing Web Search service, which can be obtained at <https://azure.microsoft.com/cs-cz/services/cognitive-services/search/>.

Apart from these every used knowledge base (the source of Linked Data resources) has to be configured separately. More in [Configuration](#).

8.1.3 Building from source files

1. Checkout the sources and accompanying resource from Git repository at <https://github.com/odalic/sti> or copy them from the installation disc.
2. Install the libraries in the `lib` directory by running the `mvninstall.bat` in the source files root directory (or in case of other OSs than Windows: running the few `mvn` commands present there manually), as these are not present in any public Maven repository.
3. Run `mvn install` in the root directory, all the sub-projects will be installed one by one and the produced `.war` will be placed in `odalic/target` subdirectory.
4. Copy the subdirectories `config` and `resources` (only these are needed during runtime) at a desired location and make the necessary changes in the main configuration file `sti.properties`, located in the `config` sub-directory and follow the rest of [Deployment section](#).

8.1.3.1 Building Javadoc documentation

The Javadoc documentation building is supported only for the `odalic` module of the server. To do that, navigate to source files repository subdirectory `odalic` and run `mvn javadoc:javadoc` to build the Javadoc files hierarchy in `odalic/target/site` or `mvn javadoc:jar` to generate a JAR packed Javadoc in the `target`. In any case, the installation disc already contains pre-generated Javadoc.

8.1.4 Security

While the application provides means to secure privileged operations according to JWT standard (*JSON Web Tokens, more in [Authentication and authorization](#)*), the requests exchanged between a client and the server are still vulnerable to eavesdropping and/or tampering. To prevent man-in-the-middle attack, we recommend using [Transport Layer Security](#) (TLS) protocol as a way of communicating with the server.

The server needs to be configured individually. If you are using Apache Tomcat, we recommend following the guidelines described on the official site <https://tomcat.apache.org/tomcat-7.0-doc/ssl-howto.html>. If you do decide to use the TLS protocol, a web client should be configured accordingly (*see [Web client installation section below](#)*).

8.1.5 Confirmation e-mails set up

In the default configuration the confirmation e-mails, which require the user to confirm the sign-up and password-changing process by following a sent hyperlink, are turned off by option `mail.confirmations = false` in the `sti.properties`. If the administrator chooses to turn them on by setting the option to `true`, he or she has to set up the other options starting with `mail`. The confirmation e-mails require an already existing running SMTP server that the application can connect to using the provided mail options such as username, password, host, SMTP port and used standard socket factory (the one using encrypted connections is in the defaults, but can be left out when unprotected connection is used). The installation files already contain tested configuration using `email.cz` as example of a common freemail provider (sans the password and login).

If the mail is not properly configured the sign-up user may not obtain the required confirmation links, thus preventing them from using the application at all.

9 Web client installation

The web client, being a single page application, works *almost* "out of the box", if a proper browser, with HTML5 support and JavaScript enabled, is used. However, to ensure the best experience, steps mentioned below should be followed:

9.1.1 Setting up a web server for the web client

The web client should reside in a web server of your choice. The server API in the default setting allows to make cross-origin requests. In order to mitigate the risk of exploitation or in case of any troubles concerning the [Same-origin policy](#) protection, modify the appropriate

headers in `CorsResponseFilter.java` from the server source, according to the location of your client instance and recompile the server. This is a good place though to remind you NOT to use or rely on Odalic in situations when your money and life would depend on it!

9.1.2 Configuring the sign-up and password-reset confirmation address on the server

Specified in detail in [Configuration](#) section, the following properties need to be correctly configured in the main configuration file, `sti.properties`, if you want to use the default web client application to respond to the confirmation URLs:

- `cz.cuni.mff.xrg.odalic.users.signup.url` - URL of the sign-up confirmation link (*received by a user on a provided e-mail address during sign-up process*)
- `cz.cuni.mff.xrg.odalic.users.reset.url` - URL of the password-reset confirmation link (*received by a user on his/her previously registered e-mail address*)

In case you are using the default web application, different web client's screens are accessible by visiting URL `.../index.html#/screenname`, `screenname` being name of the screen visited, and `index.html` being entry point of the web client application.

The token received for sign-up confirmation is processed at `.../index.html#/signup/token`, while the token for password-reset is handled at `.../index.html#/chnpasswd/token` (token being the actual token). Therefore an exemplary relevant fragment of the configuration may look like this (*we assume the web client is accessible at URL `http://localhost:8080/`, or, more specifically, at `http://localhost:8080/index.html`*):

```
# URL format for a sing-up confirmation.
cz.cuni.mff.xrg.odalic.users.signup.url=http://localhost:8080/index
.html#/signup/%s
# URL format for a password setting confirmation.
cz.cuni.mff.xrg.odalic.users.reset.url=http://localhost:8080/index.
html#/chnpasswd/%s
```

Naturally if you roll your own application based on the Odalic server, you are free to provide entirely different formatting strings for the URLs (as long as they are valid and contain the place-holder `%s` to place the generated token in).

9.1.3 Configuring address of the server in the client

Assume the `./odalic-ui/` is a path of the folder the web client application is located at.

Open `./odalic-ui/config.txt` file in a text editor. Locate the following piece of code:

```
[server]
address=http://localhost:8080/odalic/
```

Replace it with the following lines of code:

```
[server]
address=ActualServerAddress
```

Where `ActualServerAddress` is the actual server address, for example <http://localhost:8080/>.

Note that if the server uses TLS protocol as a way of securing communication with the client, `https://` should be used instead of `http://`.

9.1.4 Installing LodLive application component

The web client uses optional LodLive application component as its RDF resource browser for supported knowledge bases.

1. Let's assume the `./odalic/odalic-ui/` is a path of the folder the web client application is located at.
2. Navigate to `./odalic/` folder.
3. Copy the LodLive directory from the installation disc or checked-out from the [repository hosting the version modified to work with Odalic UI](#) and paste it in `./odalic/` folder.

9.1.5 Supported browsers

The web client can be accessed using any of the popular internet browsers with HTML5 support. Additionally, it is necessary for the browser to support [SVG](#) elements.

Some of the supported browsers are:

- Microsoft® Internet Explorer 11
- Microsoft® Edge 14 (*and newer*)
- Mozilla Firefox 50 (*and newer*)
- Google® Chrome 49 (*and newer*)

10 Plugin installation

10.1.1 Deployment

As described in [Dealing with OSGi dependency issues when DPU is imported](#), UnifiedViews plugins are packed as OSGi bundles and as such they contain their dependences in embedded JARs or declare them in their manifest. In the latter case it is the user's obligation to make them available to the chosen OSGi container. Apart from UnifiedViews version [accepting plugins of version 2.1.7](#), the Odalic plugin requires the following:

- jersey-all 2.22.2 (<http://central.maven.org/maven2/com/eclipsesource/jaxrs/jersey-all/2.22.2/jersey-all-2.22.2.jar>)
- jackson-core 2.8.6 (<http://central.maven.org/maven2/com/fasterxml/jackson/core/jackson-core/2.8.6/jackson-core-2.8.6.jar>)
- jackson-databind 2.8.6 (<http://central.maven.org/maven2/com/fasterxml/jackson/core/jackson-databind/2.8.6/jackson-databind-2.8.6.jar>)
- jackson-annotations 2.8.6 (<http://central.maven.org/maven2/com/fasterxml/jackson/core/jackson-annotations/2.8.6/jackson-annotations-2.8.6.jar>)

Copy them from the installation disc (or download them from the URLs) to lib sub-directory at the path specified with key `module.path` in the UnifiedViews frontend configuration.

Follow the instructions to create DPU Template at [DPU Template Creation](#), where the uploaded file will be the `t-odalic-{version}.jar` from the installation disc or the one built from the source files as described in the following [section](#). To set up the template, follow the instructions in the [plugin user manual](#).

10.1.2 Building from source files

1. Checkout the sources from Git repository at <https://github.com/odalic/odalic-uv-plugin> or copy them from the installation disc.
2. Run `mvn package` in the `t-odalic` directory and the produced `t-odalic-{version}.jar` OSGi plugin bundle will be placed in `t-odalic/target` directory.

11 Virtuoso installation

At least one of the knowledge bases, that are made accessible to Odalic Semantic Table Interpretation server through the associated proxies and their configuration, must be modifiable by the user. For most of the publicly accessible bases this is not the case for obvious reasons. Therefore it might be necessary for the user to establish its own knowledge base by running own [RDF store](#). Probably the most popular one is [Virtuoso](#).

11.1.1 Installation

Detailed instructions for the installation and setting-up the Virtuoso (for Windows) can be found at

<http://virtuoso.openlinksw.com/dataspace/doc/dav/wiki/Main/VOSUsageWindows#Getting%20Started%20with%20the%20VOS%20Binary%20Distribution%20for%20Windows>. The most important points are the following:

1. Download pre-compiled binaries (in an archive) from <http://virtuoso.openlinksw.com/dataspace/doc/dav/wiki/Main/VOSDownload#Pre-built%20binaries%20for%20Windows>.
2. Unpack the archive to a folder of your choice.
3. Set a system environmental variable `VIRTUOSO_HOME` to the absolute path of the folder you unpacked the archive.
4. Add the following values to the `PATH` environmental variable: `"%VIRTUOSO_HOME%/bin"` and `"VIRTUOSO_HOME%/lib"` (*usually separated by a semicolon ";"*).
5. The configuration files and Virtuoso database are now located in directory `"%VIRTUOSO_HOME%/database"`.
6. Set up Virtuoso service with the following command: `virtuoso-t +service create +instance "New Instance Name" +configfile virtuoso.ini`.
7. If you do intend to use several instances of Virtuoso (*e.g. one for DBPedia and another one for a base of different kind, e.g. Wikidata*), clone folder from `"%VIRTUOSO_HOME%/database"` and set up a Virtuoso service also for the newly cloned configuration file.
8. If you move a folder with configuration files out of its original destination, you have to correct all relative paths in the file `virtuoso.ini`.

9. Configuration parameters `NumberOfBuffers` and `MaxDirtyBuffers` should be set according to your available computer memory (*follow the instructions in the configuration file*).
10. The port number assigned to Virtuoso may be changed as well. Note that if you intend to use several instances of Virtuoso, this is an obligatory step.
11. Once a Virtuoso service is running, a web interface may be opened. (*By default it is accessible at <http://localhost:8890/conductor>*).
12. A VOS ODBC Driver should be installed as well. Download the driver and follow the instructions at <http://virtuoso.openlinksw.com/download/>.

11.1.2 Datasets import

Instructions on how to import datasets to Virtuoso may be found at <http://virtuoso.openlinksw.com/dataspace/doc/dav/wiki/Main/VirtBulkRDFLoader>. DBpedia datasets can be found at <http://wiki.dbpedia.org/Downloads2015-10>. Wikidata datasets can be found at https://www.wikidata.org/wiki/Wikidata:Database_download#RDF_dumps. Note that the import may take a significant amount of time, if a large knowledge base is used.

12 Reducing the space consumption of DBpedia

If everything (the whole DBpedia dump) is loaded to Virtuoso, it requires too much space. In order to alleviate from this burden, you can select only an essential subset of the datasets from the dump. From our experience, when you start from <http://wiki.dbpedia.org/Downloads2015-10>

dump, than the datasets most relevant to Odalic Semantic Table Interpretation are the following:

- **ontology**
- **homepages**
- **instance types**
- **labels**
- **mappingbased literals**
- **mappingbased objects**
- **short abstracts**

- **wikipedia links**

These are the Virtuoso iSQL commands you may use to load the datasets to the Virtuoso store:

```
ld_dir ('d:/KBs/DBpediaCommon', '*.owl', 'http://dbpedia.org');
ld_dir ('d:/KBs/DBpediaEN', 'homepages*.ttl',
'http://dbpedia.org');
ld_dir ('d:/KBs/DBpediaEN', 'instance_types*.ttl',
'http://dbpedia.org');
ld_dir ('d:/KBs/DBpediaEN', 'labels*.ttl', 'http://dbpedia.org');
ld_dir ('d:/KBs/DBpediaEN', 'mappingbased_literals*.ttl',
'http://dbpedia.org');
ld_dir ('d:/KBs/DBpediaEN', 'mappingbased_objects*.ttl',
'http://dbpedia.org');
ld_dir ('d:/KBs/DBpediaEN', 'short_abstracts*.ttl',
'http://dbpedia.org');
ld_dir ('d:/KBs/DBpediaEN', 'wikipedia_links*.ttl',
'http://dbpedia.org');
```

This must be confirmed by the iSQL command starting the import:

```
rdf_loader_run();
```

13 Docker image

The Odalic Docker image is available on GitHub: <https://github.com/odalic/odalic-docker>.

13.1.1 Docker setup

You can find the Docker installation files for all major platforms here:

<https://www.docker.com/community-edition>.

Please note, that Docker for Windows requires Hyper-V, so it cannot run natively on Windows 10 Home or older Windows versions. You have to use the Docker Tools instead:

<https://www.docker.com/products/docker-toolbox>.

13.1.2 Running Odalic

Odalic requires a working SPARQL endpoint with update rights. An easy way to setup such endpoint is to run the virtuoso Docker image:

```
docker run --name localKB -p 8890:8890 -e SPARQL_UPDATE=true -e
```

```
DEFAULT_GRAPH=http://odalic.eu -d tensorflow/virtuoso
```

To build the Odalic Docker image, execute:

```
docker build --tag odalic:latest https://github.com/odalic/odalic-docker.git
```

If you do not have git, you have to download the dockerfile and run:

```
docker build --tag odalic:latest .
```

The following Docker command will run the prepared Docker image:

```
docker run -ti --name odalic --link localKB -p 8080:8080  
odalic:latest
```

The Odalic UI can be now accessed from your favourite browser on localhost:8080/odalic-ui.

13.1.3 Accessing the virtual machine

If you want to access the odalic installation files and configuration, you have to run the Odalic image in a detached mode:

```
docker run -d --name odalic --link localKB -p 8080:8080  
odalic:latest
```

You can then start the Linux command line by executing:

```
docker exec -ti odalic sh
```

13.1.4 Starting and stopping the Odalic container

Running a Docker image will create a Docker container. You can start/stop the created container like this:

Code Block 3 Start

```
docker start odalic
```

Code Block 4 Stop

```
docker stop odalic
```

13.1.5 Changing ports

You can generally change a docker container port by changing the "-p" parameter:

```
-p 8080:[your local port]
```

You can freely change the local SPARQL endpoint port. It is only used when you access it directly over: <http://localhost:8890/sparql>.

To change the Odalic port, you have to also access the Odalic virtual machine and edit following files:

```
/usr/local/tomcat/webapps/odalic-ui/config.txt  
/usr/local/odalic/config/sti.properties
```

For more information please consult the Odalic documentation.

13.1.6 Updating Odalic

If you want to update your Docker container to a new Odalic version, you have to first stop the existing container. Then remove it by executing:

```
docker container rm -f odalic
```

After the container is removed, you can update the image by running the build again:

```
docker build --tag odalic:latest https://github.com/odalic/odalic-  
docker.git
```

13.1.7 Advanced Docker configuration

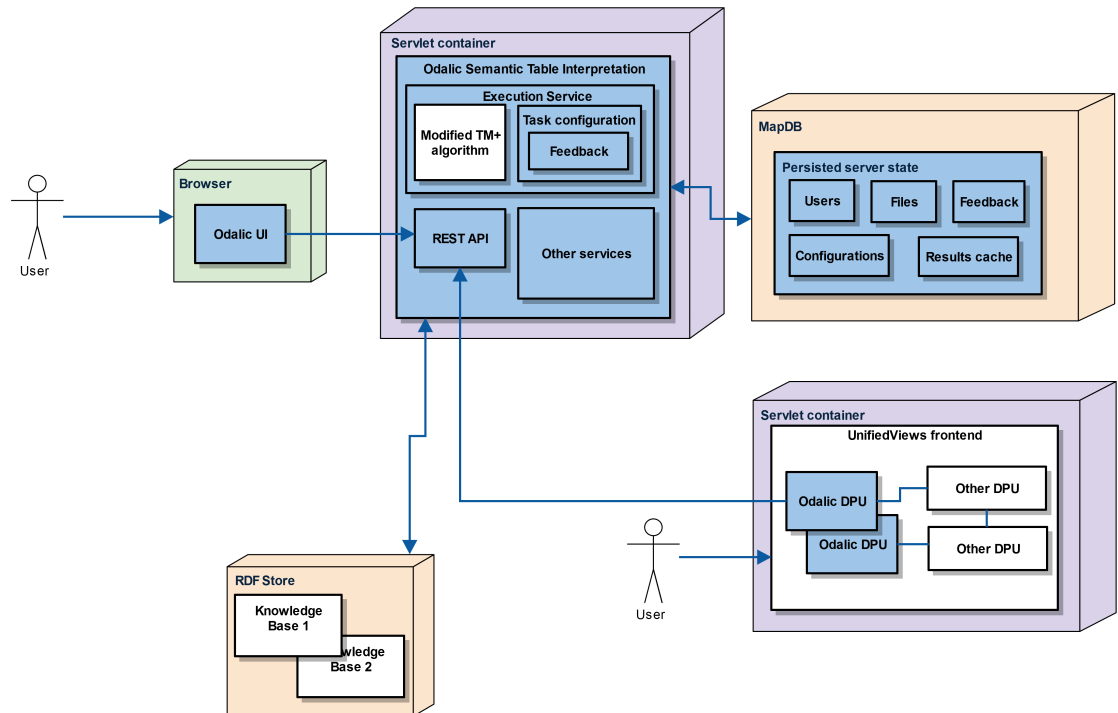
For more advanced commands please refer to the Docker documentation <https://docs.docker.com/engine/reference/commandline/cli>.

14 DEVELOPER DOCUMENTATION

- Architecture
- Server
 - Modules
 - Core algorithm description
 - REST API implementation
 - Authentication and authorization
 - Input files management and parsing
 - Tasks executions
 - User feedback
 - Result exports
 - Annotations (JSON)
 - Extended CSV
 - RDF (triples)
 - Data cube
 - Configurations export and import
 - Persisting server state
 - Configuration
 - Main Settings
 - KB Proxy Settings
 - KB Structure Settings (Groups)
 - Websearch Settings
 - RDF manipulation
- UI
- UnifiedViews DPU implementation
- Possible extensions and improvements

- Project history

15 Architecture



Odalic architecture follows the basic division into the three components: server (Odalic Semantic Table Interpretation), web client (Odalic UI) and the plugin (Odalic UnifiedViews Plugin). Both the client and server, and the plugin DPU instances and server communicate over the established server [REST API](#). Users can either use the web client to put CSV files to process, or in case they require the Odalic to integrate with other UnifiedViews DPUs or use some of the other UnifiedViews features, the plugin.

15.1.1 Odalic UI

Odalic UI is a single-page web application (*based on AngularJS*), which is running entirely in the client's browser. It uses the server REST API to let the user register, log in, send or set up files, run the processing on them, provide feedback and finally export the results or the used configuration.

In order to get notified about the change of execution state, the client actively polls the server.

15.1.2 Odalic Semantic Table Interpretation

The server is deployed in a servlet container (tested on Apache Tomcat). It runs many services, of note is the one that actually does the processing of the input files. It spawns a new thread for every set up task and passes its configuration or a collected feedback (in case it is a re-run) to algorithm which is a modified version of the TableMiner+ algorithm by Ziqi Zhang. The server responds to state polling requests, but also allows to ask for the result upfront and blocks the response to the request until the result is available.

Interesting implementation details of this and other services are described throughout the developer documentation.

15.1.2.1 Knowledge bases

Knowledge bases are hosted in a separate dedicated one or more RDF stores (e.g. [Virtuoso](#)). They can be accessed remotely, some of them are available to the general public. Access to them can be configured in `.properties` files stored in the `config` sub-directory of the working directory, which the server loads at start-up and provides to every created user. The bases can be further managed, new added and configured through the API and the UI. They are mainly used by the processing algorithm and services facilitating user feedback.

Odalic requires presence of at least one knowledge base that is modifiable, to be selected as the primary one. This base is used to store the user-defined resources and also takes precedence in case of conflict between partial results from the bases used in processing. The Docker installation comes with one such base so the user does not need to configure his or her own if he or she does not want to. This one can remain empty, only to store the proposed resources.

15.1.2.2 Server state

The server uses a [MapDB](#) file storage to keep the state of the server, which consists of the registered and logged users, their files, file and task configurations and the feedback users provided. The storage also caches execution results, so the users do not have to compute them again in case of server restart or crash, as they can be relatively expensive to do, from time and network traffic perspective.

15.1.3 Odalic DPU

Odalic DPUs work within the boundaries of [UnifiedViews architecture](#), from the Odalic platform perspective they behave as another REST API client. They even use the same authentication mechanism as the Odalic UI, where the security token is saved in the DPU template and then used in all the template instances.

16 Server

- Modules
- Core algorithm description
- REST API implementation
- Authentication and authorization
- Input files management and parsing
- Tasks executions
- User feedback
- Result exports
 - Annotations (JSON)
 - Extended CSV
 - RDF (triples)
- Data cube
- Configurations export and import
- Persisting server state
- Configuration
 - Main Settings
 - KB Proxy Settings
 - KB Structure Settings (Groups)
 - Websearch Settings
- RDF manipulation

16.1.1 Modules

All modules are located in the [sti repository of the Odalic project](#).

- **odalic**
 - This module contains almost all the extensions added to the original [STI TableMiner+](#) under this project, apart from those that modify the algorithm or work with knowledge bases and web search.

- It is divided into packages, which detailed description is provided in relevant parts of the developer documentation. They are the following (some low-level packages were left-out from this overview for sake of clarity, they are described in the Javadoc):
 - **api** (available server APIs)
 - **rdf** (RDF input and outputs - only task and bases configuration so far)
 - **rest** (REST API classes, all built atop the standard [Jersey](#) server library)
 - **bases** (list of working knowledge bases; now a full-fledged runtime bases management is supported)
 - **entities** (searching of available existing and proposing of new, custom RDF resources)
 - **feedback** (model classes used to describe the feedback user provides to the algorithm)
 - **files** (input files management and format configuration)
 - **input** (parsing of the input files and tied domain classes)
 - **outputs** (result export formats and facilitating classes)
 - **annotatedtable** (JSON-formatted annotations complementary to the extended CSVs)
 - **csvexport** (extended CSV files, see https://www.w3.org/2013/csvw/wiki/Main_Page for the standard draft)
 - **rdfexport** (RDF serialization formats)
 - **tasks** (tasks management)
 - **annotations** (classes representing the table annotations produced by the processing algorithm)
 - **configurations** (task configuration classes and service - one of the factors, next to the input and state of used knowledge bases, affecting the processing result)
 - **executions** (execution environment for the tasks, directly employing the modified TableMiner+ algorithm)
 - **feedbacks** (means to provide a feedback to the algorithm)
 - **results** (processing result representation and conversion from and to the format used by the algorithm)

- **users** (user management, authorization and authentication support)
- **util** (utilities applicable across the whole extension project)
 - **configuration** (configuration reading utilities)
 - **hash** (password hashing)
 - **mail** (e-mails delivery)
 - **storage** (keeping of the server state after shut-down or restart)
- **sti-common-utils**
 - Original STI module.
 - This module contains utility classes (cache interface, string and serialization utilities) used by more than one original sti module.
- **sti-kbproxy** (interfaces to knowledge base proxies and factory methods creating them from the configuration files or configuration provided via API)
 - **model** (model classes of the objects kept in the bases)
 - **sparql** (proxies to bases accessible through SPARQL endpoints)
- **sti-main**
 - Original STI module, stripped of non-essential parts (such as files batch processing, baseline algorithms, HTML parsing,...), cleaned-up and extended with code to accept user feedback and other improvements.
 - **core**
 - **algorithm** (the modified TableMiner+ algorithm)
 - **extension** (feedback classes mirrored from the odalic module to avoid circular dependence, their instances are fed to the algorithm, encapsulated in the Constraint instance)
 - **model** (internal annotations model of the algorithm, these classes are generally mutable and not fit to use outside of the module scope)
 - **scorer** (score computation for the candidates to annotations)
 - **subjectcol** (subject column, a.k.a "primary key" column discovery)
 - **nlp** (NLP processing tools)
 - **util**
- **sti-websearch**
 - Original STI module, but heavily rewritten in order to accommodate the new version of the Bing API.

- Contains a simple web client querying the public API of the Bing web search. Its results are used to compute scores of the candidates to annotations.

16.1.2 Core algorithm description

16.1.2.1 Phases

The main goal of the core algorithm is to take the the tabular input data, interpret it as best as it can using selected knowledge bases, and return an annotated table. This annotation process has several distinct phases:

1. **Subject Column Detection**

The subject column candidate is determined from both the column headers and cell values. The values in the subject column then represent main entities, values from other columns are interpreted as properties of these main entities.

2. **Cells Disambiguation**

Individual cell values are searched in available knowledge bases. Search results are then scored and their types and properties are loaded for further processing. The result of this phase is a list of candidates for every table cell.

3. **Columns Classification**

Classification determines candidates for columns based on types of disambiguated column cells. The desired result is to pick a class that is a type for majority of the column cells, but also is not too broad (e.g. when classifying a column of writers, we want to receive dbpedia:Writer and not dbpedia:Thing).

4. **Relations Enumeration**

Relations between cells and columns are determined from properties of disambiguated cells.

A more in-depth description of the original algorithm, upon which is the core algorithm base on, can be found in the paper <http://www.semantic-web-journal.net/content/effective-and-efficient-semantic-table-interpretation-using-tableminer-0> . Its implementation of the algorithm is spread into several modules.

- **sti-main**

The main part of the algorithm. Contains the implementation of the above mentioned steps. The original TableMiner+ contained many experimental interpreter implementations. In the Odalic, most of them were removed in favor of the TMP implementation. The reason for this is, that each of them would have to be extended by new features like user feedback, which would go beyond the scope of our project. Also there are mostly proven inferior by the paper.

- **sti-kbproxy**
Provides an abstracted interface for communication with various KBs. Used both by the core algorithm and Odalic server.
- **sti-websearch**
Handles scoring of results based on web search.

16.1.2.2 Main Module

The entry point for the algorithm is the **uk.ac.shef.dcs.sti.core.algorithm.tmp.TMPOdalicInterpreter** class, to successfully initialize the interpreter it is necessary to obtain instances of following classes.

- **uk.ac.shef.dcs.sti.core.algorithm.tmp.TCellDisambiguator**
Handles cell disambiguation.
- **uk.ac.shef.dcs.sti.core.algorithm.tmp.TColumnClassifier**
Handles columns classification.
- **uk.ac.shef.dcs.sti.core.algorithm.tmp.sampler.TContentCellRanker**
Provides ranking of rows based on number of non-empty cells. Currently used implementation is **uk.ac.shef.dcs.sti.core.algorithm.tmp.sampler.OSPD_nonEmpty**.
- **uk.ac.shef.dcs.sti.core.algorithm.tmp.LEARNING**
Performs preliminary disambiguation and classification on a sample of rows.
- **uk.ac.shef.dcs.sti.core.algorithm.tmp.UPDATE**
Updates results scores at the of phase 3, after the classification is done.
- **uk.ac.shef.dcs.sti.core.algorithm.tmp.TColumnColumnRelationEnumerator**
Discovers relations between columns and cells.
- **uk.ac.shef.dcs.sti.core.algorithm.tmp.LiteralColumnTagger**
Used at the end of phase 4. Annotates any not yet annotated columns as data properties and tries to find relations with the subject column.
- **uk.ac.shef.dcs.kbproxy**
Provides access to the underlying knowledge bases.

16.1.2.3 KB Proxy Module

The KB Proxy module is used by both Main module and Odalic server module to search configured KBs. The **uk.ac.shef.dcs.kbproxy.KBProxy** instances are created from Odalic configuration files using the **uk.ac.shef.dcs.kbproxy.KBProxyFactory**. The base class has built in Solr cache and provides methods for saving search results to the cache and retrieving them from the cache. Each KBproxy has it's own solr cache defined by the KB name. There are currently two implementations of the **uk.ac.shef.dcs.kbproxy.KBProxy**.

- **uk.ac.shef.dcs.kbproxy.sparql.SPARQLProxy**
Generic implementation of KB Search of SPARQL KBs.
- **uk.ac.shef.dcs.kbproxy.sparql.DBpediaProxy**
Specific implementation of KB Search for DBpedia type of KBs. Extends the **SPARQLProxy** and currently has only modified label retrieval methods.

The original TableMiner+ used proxy class for Freebase. This was replaced by a more generic SPARQL proxy. The Freebase is no longer supported, because it's original public API is no longer available. The **uk.ac.shef.dcs.kbproxy.KBProxy** has four main groups of public methods.

Core algorithm search

These methods are used by the core algorithm, they do not throw any exceptions. To implement them, it is necessary to override the "***Internal**" methods with same names. Any potential errors are caught and returned as warning for the user.

- **findAttributesOfClazz**
Returns a collection of attributes of the selected class.
- **findAttributesOfEntities**
Returns a collection of attributes of the selected entity.
- **findAttributesOfProperty**
Returns a collection of attributes of the selected property.
- **findEntityCandidates**
Method used for the preliminary disambiguation or for main disambiguation when the preliminary disambiguation returned no types. Searches for candidates in the KB based on their label. The entities are returned with complete information about attributes and types.
- **findEntityCandidatesOfTypes**
Same as **findEntityCandidates** with the difference that results are only of certain types. Used in disambiguation when preliminary disambiguation returned some candidate types.
- **findEntityClazzSimilarity**
Evaluates similarity between two classes.
- **findGranularityOfClazz**
Evaluates granularity of a class.
- **loadEntity**
Loads single entity from the KB with complete information about attributes and types.

User initiated Search

These methods are used by the Odalic server in the user search dialog.

- **findPredicateByFulltext**
Returns candidate entities (predicates) from the KB based on supplied string value, domain and range.
- **findResourceByFulltext**
Returns candidate entities (resources) from the KB based on supplied string value.
- **findClassByFulltext**
Returns candidate entities (classes) from the KB based on supplied string value.

Proposals

- **isInsertSupported**
Information about whether the knowledge base supports inserting new concepts.
- **insertClass**
Inserts a new class into the knowledge base.
- **insertConcept**
Inserts a new concept into the knowledge base.
- **insertProperty**
Inserts a new property type into the knowledge base.

Export

- **getPropertyDomains**
Returns domain of the given resource.
- **getPropertyRanges**
Returns range properties of the given resource.

SPARQL Proxy

The SPARQL Proxy implements the above mentioned methods using [Jena](#) to generate the required SPARQL SELECT, INSERT and ASK requests. The original TableMiner+ created SPARQL queries by string concatenation, the current approach makes use of Jena "builder" classes and is both more readable and less error prone. The fulltext search is implemented by querying the DBpedia fulltext catalogue through the "**bif:contains**" predicate. If the fulltext catalogue is not available, the proxy falls back on regex based filters, that are somewhat slower.

In user search, it is important to be able to return any resources found during the disambiguation. This is not always possible, because some results, like types from column classification, may not have labels. It is also important to be able to find any recently proposed resources. For this reason, the user initiated search always performs both exact match query and fulltext query. The exact match query usually returns less results, but can find recently

proposed resources, that have not yet been added to the fulltext catalogue of the knowledge base.

The disambiguation of cells usually creates following queries.

1. Exact match query by label

Code Block 5 Example

```
PREFIX geonames: <http://www.geonames.org/ontology#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
... # Other prefixes left out.

SELECT DISTINCT ?subject
WHERE
  { { SELECT DISTINCT ?subject
      WHERE
        { { ?subject foaf:name "Gardens of the Moon"@en }
          UNION
            { ?subject dbpprop:fullname "Gardens of the
Moon"@en }
          UNION
            { ?subject rdfs:label "Gardens of the Moon"@en }
          UNION
            { ?subject dbpprop:name "Gardens of the
Moon"@en }
          UNION
            { ?subject dbpedia-owl:originalTitle "Gardens of
the Moon"@en }
        }
      }
    ?subject rdf:type ?class
  }
```

2. Fulltext query by label

Code Block 6 Example

```
# Prefixes left out.

SELECT DISTINCT ?subject ?object
WHERE
  { { SELECT DISTINCT ?subject ?object
      WHERE
        { { ?subject foaf:name ?object .
          ?object <bif:contains> "\"Gardens\" AND \"of\"
AND \"the\" AND \"Moon\""}
        }
      UNION
        { ?subject dbpprop:fullname ?object .
          ?object <bif:contains> "\"Gardens\"
AND \"of\" AND \"the\" AND \"Moon\""}
        }
      UNION
        { ?subject rdfs:label ?object .
          ?object <bif:contains> "\"Gardens\" AND \"of\"
AND \"the\" AND \"Moon\""}
        }
      UNION
        { ?subject dbpprop:name ?object .
          ?object <bif:contains> "\"Gardens\" AND \"of\"
AND \"the\" AND \"Moon\""}
        }
      UNION
        { ?subject dbpedia-owl:originalTitle ?object .
          ?object <bif:contains> "\"Gardens\"
AND \"of\" AND \"the\" AND \"Moon\""}
        }
    }
  }
  ?subject rdf:type ?class
}
```

3. Query attributes for every result

Code Block 7 Example

```
# Prefixes left out.

SELECT DISTINCT ?predicate ?object
WHERE
  { dbpedia:Gardens_of_the_Moon
    ?predicate ?object
  }
}
```

4. Query label for every attribute

Code Block 8 Example

```
# Prefixes left out.

SELECT DISTINCT ?object
WHERE
{
  { dbpedia-owl:Book
    foaf:name ?object
  }
  UNION
  { dbpedia-owl:Book
    dbpprop:fullname ?object
  }
  UNION
  { dbpedia-owl:Book
    rdfs:label ?object
  }
  UNION
  { dbpedia-owl:Book
    dbpprop:name ?object
  }
  UNION
  { dbpedia-owl:Book
    dbpedia-owl:originalTitle ?object
  }
}
```

16.1.2.4 Websearch Module

Provides functionality for searching concepts of the web. Used for scoring results. Formerly implemented using Bing Web Search API, now renamed to Microsoft Cognitive Services.

16.1.3 REST API implementation

16.1.3.1 Mapping

We opt for [Jersey](#) to implement the [specified REST resources](#). As the reference JAX-RS implementation it is well supported, extensible and blends good enough with the Spring container. Considering that the first client of the API is to be the Odalic UI, AngularJS web application, choosing JSON as the primary data format was reasonable as well. When choosing how to map the domain objects to JSON, two opposing concerns emerged:

1. ease of mapping directly the domain objects by marking them with JAXB annotations (work for JSON too, but even then require certain concessions to the classes design) on one hand,

2. and the need to build a robust domain model that would fare well when its objects would be passed to the algorithm as feedback, or its classes used when adding new functionality.

Although JAXB allows to annotate the class fields and make the constructor private, and there is no need to make the mapped class to be a true Java Bean, this still does not make its instances reliable to use within the domain model: fields missing in the original JSON would still be initialized to null, so even than the domain class would have to validated before use. And that makes the domain model too fragile from the beginning.

In the end a solution separating the domain classes (used throughout the project code) and their counterparts (located in `cz.cuni.mff.xrg.odalic.api.rest.values`), that would be mapped to JSON, was chosen. Thanks to [XmlAdapters](#) most of the domain classes can be seamlessly converted to the mapped ones and back, and it is even possible to use them in the method signatures that handle HTTP requests, making the mapping almost invisible to the rest of the server code. This solution has some drawbacks, mainly It proved to be next to impossible to inject dependencies from Spring container to XmlAdapters. For example an XmlAdapter cannot be used to convert from Task JSON value to Task domain object, if you want to make sure that the file referred by task exists. In this case you have to manually convert from the `cz.cuni.mff.xrg.odalic.api.rest.values.TaskValue` to `cz.cuni.mff.xrg.odalic.tasks.Task`. in the Jersey resource class, where the file service can be injected without a problem. On the other hand there are some perks:

- XmlAdapters are resolved recursively on the whole class hierarchy, so if you adapt a domain class that contains fields pointing to instances of other domain classes, you do not have to change the type of the fields in the mapped version, their XmlAdapters will be used automatically.
- The adapters work well alongside custom JSON serializers and de-serializers from [jackson-databind](#) library. This allows to convert fields of complex type, e.g. nested Maps with non-String keys. This helped to encode a feedback on individual, sparsely distributed table cells in a relatively compact way (see `cz.cuni.mff.xrg.odalic.api.rest.values.DisambiguationValue` and compare with the output produced, as seen in [in the REST API specification examples](#)).

16.1.3.2 Other features

The API implementation relies on other useful Jersey features, such as request and response [filters](#) and [exception mappers](#). The filters are used to set [CORS](#) headers, log the requests and to facilitate [authorization and authentication](#). An exception mapper was used to convert the exceptions thrown during the course of request processing to JSON messages presented in the specification. Concerning the overall exception handling strategy, we chose to strictly separate the API from the underlying services and vowed not to raise web application exceptions in code that is not aware of being executed in web application. This required responsible mapping from general exceptions raised by the implementing services to the

`WebApplicationException`s, that were in the end turned by the mapper into appropriate HTTP responses.

16.1.4 Authentication and authorization

16.1.4.1 From a web client perspective

The web client uses Satellizer library (<https://github.com/sahat/satellizer>) to support server security demands where the parties agreed on use of tokens adhering to the `JWT` standard.

Signing up

1. User creates a new account by providing relevant information, such as e-mail, by which his or her account will be identified.
2. The web client sends a request to the server with the relevant information (*e-mail, password*).
3. In case no other account is paired with the provided e-mail address, the server creates the new account and responds with an affirmative message. The server additionally asynchronously sends an e-mail to the provided address.
4. The web client informs the user to await an e-mail with additional instructions.
5. The e-mail consists (*besides human readable instructions*) of a hyperlink leading to a specific URL, as configured in the previously mentioned `config/sti.properties` file. For example, `http://localhost:8080/index.html#/signup/<token>`, where "`<token>`" is an actual token provided by the server.
6. By clicking on the link, the user is redirected to a specific screen on the web client. There, the application automatically parses the URL and sends a confirmation request to the server with the token attached.
7. If the token has not expired yet, the server responds with an affirmative message, to which the client application responds by displaying information to the user, ensuring him or her the account has been successfully activated, and requesting him or her to log in.

Logging in

1. The user logs in by providing the e-mail address his or her account is now associated with, and the password.
2. The application sends a request to the server containing the e-mail and the password.
3. If the account information is valid, the server responds with an affirmative message with a token attached.

4. The application saves the token in the user's local storage (*by using HTML5 Web Storage API*).
5. **From now on, each request the application sends is automatically attached with an authorization header containing the token.**
6. If the user logs out, the token is removed from the local storage.
7. A server may respond to a request with a negative message, even if the token is provided in the authorization header of the request (*e.g. when the token is expired*). In that case, the application redirects the user to "log in" screen, where it automatically re-evaluates the validity of the token (*by sending a testing request to the server*). If the token is not valid anymore, the user is automatically logged out and the token removed from the local storage. The user may log in again.

Because the communication between the web client and the server is still vulnerable to the "man-in-the-middle" attacks, a usage of TLS/SSL communication protocol is highly recommended.

Also the confirmation emails can be turned off in the main configuration.

16.1.4.2 From the server perspective

Tokens

Employing [JWT](#) tokens to authenticate the client of the REST API has several appealing properties, such as:

- They can be shared with 3rd parties to enable them to act on behalf of the user without compromising the actual credentials (e-mail and password combination).
 - Such is the case of the Odalic UnifiedViews plugin.
- They can carry useful information which does not have to be remembered by the server, but still can be relied on.
 - But it is good to keep in mind that unless other measures are taken, these pieces of information are publicly visible!
 - For example Odalic server stores the date of expiration in the tokens, which is checked by the server, without the fear that it has been tampered with.
- Unless they are long-lived enough to endanger the user, the server does not have to remember which tokens were issued.
 - Unfortunately the processing of tasks may take a long time, during which the client polls the server and it would be impractical to negotiate a new token. So the Odalic server does remember which tokens were issued (their ID is enough). Only then it

can revoke them when needed, which is implemented by letting the user change his or her password.

- They can be safely shared in a URL, unless they are excessively long, because they are composed from safe characters.
 - This is useful in the case of e-mail confirmation, because the token can be placed in the link itself.
- They are well supported across languages and platforms.

The server uses the [auth0 Java JWT library](#) to issue and verify sign-up and password reset tokens and the tokens authenticating the source of the API calls. As the available memory is always limited, only a certain amount of tokens is remembered by the server, when this limit is reached, the oldest issued token is lost.

Authentication and authorization with Jersey

To secure REST endpoints, the privileged [Resource](#) classes or their methods (if higher granularity is needed) are marked with `cz.cuni.mff.xrg.odalic.api.rest.Secured` annotation, which triggers checking code in Authentication request filter. This code calls upon the `cz.cuni.mff.xrg.odalic.users.UserService` to match the token received in HTTP Authorization header with some of the valid issued ones. The `Secured` annotation also has a parameter which allows to specify which role the user must have to access the secured endpoint. This is handled by Authorization request filter which compares the callee's role with the set of acceptable ones. Should any of these checks fail, an exception is thrown and mapped to appropriate HTTP response, thus denying the access. So far only two roles are supported: common users and the administrator. The administrator has all the rights of the regular users, plus it can access endpoints allowing to list all the users and delete some of them. Also he or she can act on behalf of another user by providing his or her user ID in the calls.

Password handling

The password set during the sign-up or reset is never stored. The server only computes a hash from it and the assigned salt and compares it with a password sent in login credentials. For this an [Scrypt implementation in pure Java](#) was used. It may lack the raw performance, but does not need to be linked to a C library.

16.1.5 Input files management and parsing

Odalic server distinguishes between two kinds of input CSV files: local and remote ones. Local files get to be uploaded by the user and reside in the server storage. Remote ones are defined only by their URL. Both kinds share the same structure for meta-data (differing only in their cached flag), and even the local ones have their URL assigned, which is the same under which they can be GET from the REST API. The files are parsed only moment before the execution, so it is possible to get different results if the underlying remote files change or the parsing format of both the local and remote file is changed by the user. Every input file can be shared among multiple processing tasks belonging to the same user, therefore it cannot be deleted as long as at least one task refers to it. The references are kept by the implementing FileService, where for each referring Task has to be subscribed and unsubscribed upon deletion.

Parsing is done through [Apache Commons CSV library](#), which is able to detect line separators, but provides no mean to obtain that information for further use. So detecting code was added to the parser, because the used line separators are needed in order to export the results in the form expected by the client. The result of parsing is a model of CSV file represented by `cz.cuni.mff.xrg.odalic.input.Input`, consisting of the the list of rows and header (first row). Odalic assumes the file to be consistent, that means containing equal number of records in all rows, and raise an exception if this is not the case.

16.1.6 Tasks executions

16.1.6.1 Execution workflow

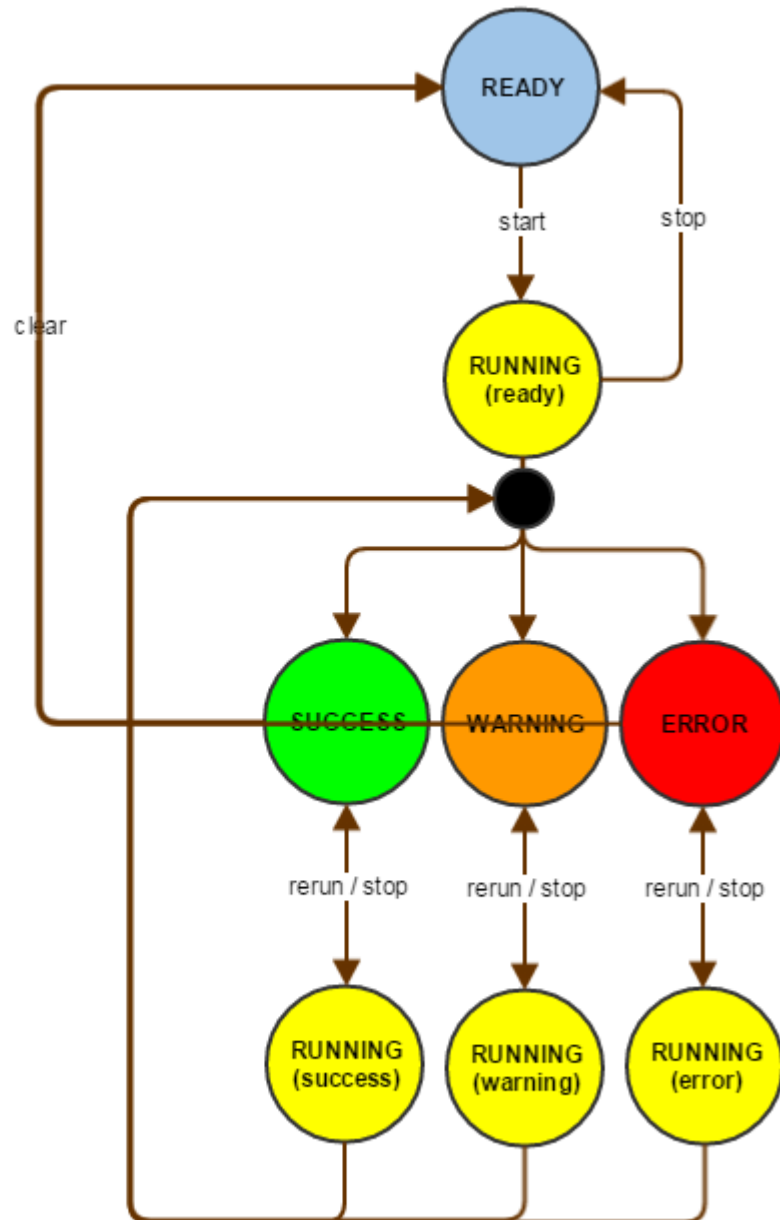
When the user gives command to execute a processing task, the following happens:

1. Server verifies that execution is not already in process.
 - To simplify matters we allow only one execution per task definition. This is only a slight practical limitation as nothing prevents the user from copying the exact task definition under another name (via export/import round-trip).
2. The current [configuration](#) is retrieved.
3. A snapshot of the the parsed input specified in the configuration is made.
 - This prevents possible corruption of the result display in case the formatting of the original file changes between the start of processing and the result presentation. This might cause columns mismatch and other undesirable effects.

- The parser also automatically detects line separators and remembers them to apply in outputs.
4. A Runnable instance is created, which does the following:
 - a. The parsed input is converted to the working table format of the core algorithm.
 - b. Interpreters associated with the defined knowledge base proxies are provided with the table and started one by one (although parallel computation is about to be explored in future releases), each applying the algorithm on the table, constrained by the optionally provided user feedback from previous runs.
 - c. Their results are merged into single result which is cached and set as a [Future](#) result, waiting for retrieval by the client or returning immediately if the retrieval call has already been made.
 5. The Runnable instance is submitted to [ExecutorService](#) and the returned Future is associated with the user's ID and task ID, but not before the previously cached result is purged.

When the command to cancel the task is received, it is passed on the Future. Unfortunately in the current version it is not possible to actually stop the execution and it always has to go through even if it does appear stopped the client code. At least until cooperative handling of thread interruption is introduced to the algorithm code. The problem lies in the Solr caches, which do not handle the pre-emptive interruptions well and are in the risk of corruption if the client code does them.

16.1.6.2 Execution states



The state of the Future (whether it is done, and if so, if it is cancelled) and whether there is even any associated with the task, or a result is cached, determines the possible execution states of the task. It is the caching in persistent storage, that slightly complicates things against the memory-only version, because the Futures naturally cannot be serialized, and any exceptions raised during the task processing even less so. This causes the ERROR state to be

lost upon server restart, which was ultimately deemed an acceptable result, compared with the effort needed to keep the arbitrary error cause serialized. Naturally the task result is available only when the execution is in the SUCCESS or WARNING state.

The warnings presented when querying the result from the WARNING state are collected from minor runtime exceptions that occur throughout the algorithm execution. The motivation for their introduction was the fact that in the original algorithm version even minor setbacks, like temporary HTTP connection problems, caused the entire processing to halt, which was frustrating in cases when the processing took several-hours to repeat.

16.1.6.3 Task processing result

Odalic internal model of table annotations (`cz.cuni.mff.xrg.odalic.tasks.results.Result`), presented on the outside by the REST API, is geared towards making the working of the client easier first, rather than truly reflect any theoretical model. It has the following parts:

- **subjectColumnPositions**
 - mapping of knowledge bases to subject columns
 - subject columns (concept similar to a primary key, all the relations lead from them) are determined through the run of the algorithm for each base in its own interpreter
- **headerAnnotations**
 - each column gets assigned an [RDF class](#) per knowledge base and that implies that all the cells in the column are its instances
 - Actually there are usually multiples candidates, which are ordered according to their score and left for the user to choose from, if not satisfied with the default choice.
- **cellAnnotations**
 - A matrix of RDF resources, one per each cell at most, that disambiguate the original cell content.
 - The format of the cell annotations is actually quite similar to those in headers, as there is not much need to distinguish between them for most uses.
- **columnRelationAnnotations**
 - RDF properties forming a relation between two columns.
 - The domain of the property conforms to the class of the first column, the range to the second one.
- **statisticalAnnotations**

- They assign set of property to each column per base and a type indicating whether the column is a measure of dimension of the formed data cube.
- Applies only when a statistical processing was chosen during the task configuration. This basically means that no relations were formed, and the choice of subject column was also dropped, because what matters is the final data cube formed during the export.
- **columnProcessingAnnotations**
 - Explicit information whether the column was declared by the algorithm to contain named entities (most of the disambiguated content), non-named entities (typically numerical values) or if the column was ignored by the algorithm (on the user's request) during the process.
- **warnings**
 - List of warnings in the order as they were written when non-fatal errors occurred during the processing.

These parts are adapted from the filled by adapting the result representation provided by the algorithm (`uk.ac.shef.dcs.sti.core.model.TAnnotation`), which by itself is not suitable to be passed around, but designed to be modified during the processing.

Annotation structure

Every annotation (for headers, cells, relations,...) has the following structure:

- Map from knowledge bases to ordered set of candidates.
 - Every candidate is composed from the actual resources description and score assigned by the algorithm.
 - Resource description has a label and the resource URI (optionally prefixed as is common in many common [representations](#)).
- Map from knowledge bases to the set of chosen candidates.

16.1.7 User feedback

User can provide several kinds of feedback to the core annotation algorithm. All parts are grouped in the `cz.cuni.mff.xrg.odalic.feedbacks.Feedback` class. Feedback is a part of Task configuration. Initially (after the Task creation) the feedback is empty. When the algorithm run is finished, user can observe the results and adjust the feedback, which will be considered by algorithm in the next run.

16.1.7.1 Passing the feedback to the algorithm

Before the execution of the core algorithm, Feedback is adapted by the `cz.cuni.mff.xrg.odalic.feedbacks.DefaultFeedbackToConstraintsAdapter` (which implements interface `cz.cuni.mff.xrg.odalic.feedbacks.FeedbackToConstraintsAdapter`) to object of the `uk.ac.shef.dcs.sti.core.extension.constraints.Constraints` class, which is the extension of `sti-main` module and encapsulates user's suggestions for concrete Semantic table interpreter run. The difference is, that Feedback comprises suggestions for all used knowledge bases, but Constraints contain suggestions only for one knowledge base, which is actually used in particular interpreter run.

Each run of the core algorithm is processed by the `TAnnotation` `uk.ac.shef.dcs.sti.core.algorithm.tmp.TMPOdalicInterpreter.start(Table table, boolean statistical, Constraints constraints)` method, which originally did not accept feedbacks, so we added (and implemented) a new argument `constraints` to this method. We also added another argument `statistical`, which does not come from Feedback, but from Task configuration. The statistical data annotations part of the Feedback is considered only when the `statistical` boolean argument is set to `true`.

16.1.7.2 Subject columns positions

User can suggest the positions of the subject columns, which then serve as the subjects for the relation discovery process. The original TableMiner+ algorithm detected the main subject column in the `List<Pair<Integer, Pair<Double, Boolean>>>` `uk.ac.shef.dcs.sti.core.subjectcol.SubjectColumnDetector.compute(Table table, int... skipColumns)` method according to the column data-types and other features and did not support manual (user's) setting of the subject columns. Relations were originally discovered only for the subject column detected by that method. Now when the subject columns are suggested by the user, relations are discovered for all of them.

16.1.7.3 Ignoring the column

User may want to completely ignore the column, so that the column is not annotated (disambiguated/classified) at all and also not considered in relations. This functionality was already supported in the original algorithm and used in all its phases, but we changed the way of passing the information about the indices of columns, which the user wants to ignore, to the algorithm. Originally the indices were set in the configuration file and passed as argument to the constructor of the `uk.ac.shef.dcs.sti.core.algorithm.SemanticTableInterpreter` object (or inherited class objects). In our implementation we store the column positions in the Feedback (resp. Constraints) class and pass as argument to the `start` method of the `TMPOdalicInterpreter` (introduced above). This approach was chosen, because the information about ignored column is part of the same object as other kinds of feedback and can be set individually for each algorithm run.

16.1.7.4 Compulsory columns

The algorithm performs classification and disambiguation phase only for columns where the cells' content data type was recognized as named entity. User may want to compulsorily perform them also when the content does not seem as named entity (e.g. when there are numerical identifiers). This functionality was already supported in the original algorithm, but we changed the way of passing the information about the indices of compulsory columns. Originally the indices was set in the configuration file and passed to the constructor of the `uk.ac.shef.dcs.sti.core.algorithm.SemanticTableInterpreter` object (or inherited classes). We store the positions in the Feedback (resp. Constraints) and pass as argument to the `start` method of the `TMPOdalicInterpreter` (the same way as for ignored columns).

16.1.7.5 Ambiguity

User can leave certain cell of the input file ambiguous, meaning that the cell is not disambiguated. The functionality of skipping the cells was implemented in the original algorithm and used in the learning phase in the `void uk.ac.shef.dcs.sti.core.algorithm.tmp.LEARNING.learn(Table table, TAnnotation tableAnnotation, int column, Constraints constraints)` method, but not utilized in the experimental batches. So we added new argument `constraints` to this method and the positions of cells, which the user wants to skip, are read from this argument. The reading is provided by the `Set<Integer> uk.ac.shef.dcs.sti.core.extension.constraints.Constraints.getSkipRowsForColumn(int columnIndex, int rowCount)` method.

16.1.7.6 ColumnAmbiguity

This is just a short-cut for leaving all cells in the column ambiguous. It is manifested in the `Set<Integer> uk.ac.shef.dcs.sti.core.extension.constraints.Constraints.getSkipRowsForColumn(int columnIndex, int rowCount)` method, which returns all the rows to skip when the `ColumnAmbiguity` is set for given column.

16.1.7.7 Classification

User can set custom classification resource for the column. In the original algorithm the classification resource is voted according to classes of disambiguated cell values in the `Pair<Integer, List<List<Integer>>>` `uk.ac.shef.dcs.sti.core.algorithm.tmp.LEARNINGPreliminaryColumnClassifier.runPreliminaryColumnClassifier(Table table, TAnnotation tableAnnotation, int column, Constraints constraints, Integer... skipRows)` method. So we added new argument `constraints` to this method, and that extension allowed us to skip the process of voting and just set the annotation chosen by the user. When the user explicitly sets empty classification, then the column is left unclassified. Suggested classifications also affect the process of

disambiguation, because in that case the candidates for disambiguation are searched in the knowledge base with the type restriction of respective classification. But when the list of candidates searched with this type restriction is empty, then the candidates are searched again without the type restriction.

16.1.7.8 Disambiguation

User can set custom disambiguation resource for the cell. In the original algorithm the candidate entities are searched in the knowledge base according to the cell value in the void `uk.ac.shef.dcs.sti.core.algorithm.tmp.LEARNINGPreliminaryDisamb.runPreliminaryDisamb(int stopPointByPreColumnClassifier, List<List<Integer>> ranking, Table table, TAnnotation tableAnnotation, int column, Constraints constraints, Integer... skipRows)` method. So we added new argument `constraints` to this method. In that extension we also search in the knowledge base, because we need to fetch other attributes of the entity, but we search for just one entity defined by its URI, so the searching is faster and simpler. This also means, that when the user sets entity which does not exist in the knowledge base, we can warn him (in the Warnings part of the Result). Then the entity suggested by user is set as a candidate to the disambiguation algorithm in the same way as it was originally (which was searched according to the cell value) to compute scores. This means that when the suggested entity is not suitable for the cell value disambiguation, it can have the score value of zero, When the user explicitly sets empty disambiguation, the cell is left ambiguous.

16.1.7.9 Relations

User can set custom predicates for relations between columns. In the original algorithm the relation enumeration phase is provided by the void `uk.ac.shef.dcs.sti.core.algorithm.tmp.RELATIONENUMERATION.enumerate(List<Pair<Integer, Pair<Double, Boolean>>> subjectColCandidadteScores, Set<Integer> ignoreCols, TColumnColumnRelationEnumerator relationEnumerator, TAnnotation tableAnnotations, Table table, List<Integer> annotatedColumns, UPDATE update, Constraints constraints)` method. So we added new argument `constraints` to this method. In that extension we just add the column relation annotations suggested by user. This does not restrict the rest of relation discovery process, which can discover relations also between other columns (with consideration of subject column) than the user suggested.

16.1.7.10 Statistical data annotations

This is our completely new extension. In our implementation the user can denote (in the Task configuration) that the statistical data are processed. In this case the relation enumeration phase is skipped (because columns describe dimensions and measures of the statistical data cube and there are not relations in the strict sense between them). Instead of relations, the statistical annotations are set in the void `uk.ac.shef.dcs.sti.core.algorithm.tmp.TMPOdalicInterpreter.setStatisticalAnnotations(List<Integer> annotatedColumns, Table table, TAnnotation tableAnnotations,`

constraints constraints) method. Initially the algorithm considers named entity columns as dimensions and non-named entity columns as measures. The user can change this later. Then the user also (manually) sets the predicates which correspond to the columns (dimensions and measures). These annotations are used for RDF export, which is generated according to the RDF data cube standard, where each row of the input file represents one observation and predicates describe dimensions and measures of the observation.

16.1.8 Result exports

User can export results of semantic table annotation in three forms:

- [Annotations \(JSON\)](#)
- [Extended CSV](#)
- [RDF \(triples\)](#)

All classes providing the export are located in the `cz.cuni.mff.xrg.odalic.outputs` package (resp. its subpackages).

16.1.8.1 Annotations (JSON)

This file represents annotations to the extended CSV file according to the [CSV on the Web](#) standard. It contains metadata for each of "physical" columns of the CSV file and also for extra "virtual" columns describing classifications, relations or statistical data cube.

Model of Annotated table

AnnotatedTable
TableContext context
String url
TableSchema tableSchema
TableContext
String csvw
Map<String, String> mapping
TableSchema
Collection<TableColumn> columns

TableColumn
String name
Collection<String> titles
String description
String dataType
boolean virtual
boolean suppressOutput
String aboutUrl
String separator
String propertyUrl
String valueUrl

Example

```
{
"@context": "http://www.w3.org/ns/csvw",
"url": "file.csv",

"tableSchema": { "columns": [
{ "name": "City", "titles": [ "City", "LAU_NAME", "Town" ],
"dc:description": "City of Austria", "datatype": "string"
  "aboutUrl": "{City_url}",
  "propertyUrl": "dcterms:title"},
{ "name": "District", "titles": [ "City", "DISTRICT_NAME" ],
"dc:description": "District of Austria", "datatype": "string"
  "aboutUrl": "{District_url}",
  "propertyUrl": "dcterms:title"},
{ "name": "POP_FOR_NAT", "titles": [ "Population of foreigners" ],
"dc:description": "Population of foreigners in the given area",
"datatype": "string"
  "aboutUrl": "{City_url}",
  "propertyUrl": "x:hasPopulationNat"},
{ "name": "POP_TOTAL", "titles": [ "Population Total" ],
"dc:description": "Total Population in the given area", "datatype":
"string"
  "aboutUrl": "{City_url}",
  "propertyUrl": "x:hasPopulationTotal"},

{ "name": "City_type",
  "virtual": true,
  "aboutUrl": "{City_url}", //relies on value within the column
City - does it take valueUrl automatically (if available)?
  "propertyUrl": "rdf:type",
  "valueUrl": "http://adequate.at/concept/city"},

{ "name": "City_url", //do not produce to the output JSON, this
is used only to hold identifier for the row/part of the row
  "suppressOutput" : "true",
  "datatype": "anyURI"
  "valueUrl": "{City_url}"

{ "name": "City_alternative_urls", //in the form
"http://example.org/1 http://example.org/2"
  "aboutUrl": "{City_url}", //relies on value within the column
City - does it take valueUrl automatically (if available)?
  "separator": " ",
  "propertyUrl": "owl:sameAs",
  "valueUrl": "{City_alternative_urls}"},

{ "name": "District_type",
```

```

    "virtual": true,
    "aboutUrl": "{District_url}",
    "propertyUrl": "rdf:type",
    "valueUrl": "http://adequate.at/concept/district"},},

{ "name": "District_url",
  "suppressOutput" : "true",
  "datatype": "anyURI"
  "valueUrl": "{District_url}, },

{ "name": "City_District_liesIn",
  "virtual": true,
  "aboutUrl": "{City_url}",
  "propertyUrl": "ad:liesIn",
  "valueUrl": "{District_url}"}, },
}, //end of table schema
} //end

```

Export process

First the annotations for original columns are created. They include original column name (also as title), dataType String and when the column is disambiguated, then the relation between corresponding *_url column and the column name with predicate dcterms:title is added. When the disambiguations exist, the annotations for *_url (which only holds the URI of disambiguation), resp. *_alternative_urls (which describes also relation between *_url and *_alternative_urls with predicate owl:sameAs) are also created.

Then the header annotations (classifications) from the Result are read and for each of them the virtual column describing the relation between *_url and classification resource with predicate rdf:type is created.

Finally the column relation annotations from the Result are read and corresponding virtual columns are created. The subject must be always resource. But the object can be also literal (for example string or number). In this case we must also find the object dataType, which is derived from the Range property of the relation predicate entity.

17 Statistical data specifics

When statistical data are processed, some more columns (apart from columns with original input values, columns with disambiguation and alternative disambiguation URLs and virtual columns describing classifications) are added to the Annotated table JSON (virtual columns describing relations are not present, because relation discovery part of the algorithm is skipped). Some newly added virtual columns describe just one certain triple (without links to columns), because it is needed for definition of data cube.

When the URL is written in prefixed form (compact IRI), the appropriate prefix mapping is added to the "@context" attribute of annotated table. The "@context" attribute then contains array with two objects: first item is String with link to the definition of CSV on the Web standard context, second item is a map with prefixes used in the document (local context).

Dataset definition includes three virtual column with concrete triples: {datasetUri} rdf:type qb:DataSet , {datasetUri} dcterms:title {inputIdentifier} and {datasetUri} qb:structure {dsdUri} , where datasetUri is generated according to the template

"{kbInsertSchemaElementPrefix}dataset/{UUID}" and dsdUri according to

"{kbInsertSchemaElementPrefix}dsd/{UUID}" , inputIdentifier is set by user during file upload and kbInsertSchemaElementPrefix is fetched from the KnowledgeBase configuration (property called kb.insert.prefix.schema.element).

Data structure definition includes one virtual column with triple: {dsdUri} rdf:type qb:DataStructureDefinition.

Then we add column "OBSERVATION_url" as holder for URLs of observations (similar as other "*_url" columns for URLs of disambiguations) and two virtual columns linking the "OBSERVATION_url" with rdf:type predicate to qb:Observation and with qb:dataset predicate to {datasetUri}.

Component definition includes two virtual columns with concrete triples: {dsdUri} qb:component {compUri} and {compUri} {kind} {colPredicate} , where compUri is generated according to template "{kbInsertSchemaElementPrefix}dimension/{UUID}" in case of dimension component or "{kbInsertSchemaElementPrefix}measure/{UUID}" in case of measure component, kind is qb:dimension for dimension or qb:measure for measure and colPredicate is set by user in feedback (predicate describing concrete relation of dimension or measure associated with column of input file).

Next part is slightly different for dimension and for measure. Dimension definition includes triples: {colPredicate} rdf:type rdf:Property , {colPredicate} rdf:type qb:DimensionProperty , {colPredicate} rdfs:label {colPredicateLabel} and {colPredicate} qb:concept {colClassification} , where colPredicateLabel is label associated with colPredicate entity and colClassification is entity used for classification of the column. Measure definition includes triples: {colPredicate} rdf:type rdf:Property , {colPredicate} rdf:type qb:MeasureProperty , {colPredicate} rdfs:label {colPredicateLabel}, {colPredicate} qb:concept {colClassification} and {colPredicate} rdfs:subPropertyOf sdmx-measure:obsValue .

And finally there is a virtual column describing relation between OBSERVATION_url column and the column which is associated with the component. Relation predicate is colPredicate. In case of dimension component the link points to the *_url column with disambiguation URLs. For measure component the link points to the original column, which is not disambiguated, because it does not contain named entity.

17.1.1.1 Extended CSV

We generate the extended CSV file from the original input file by adding extra columns for the disambiguated entities. These extra columns are named *_url for disambiguations from primary knowledge base and *_alternative_urls for disambiguations from other knowledge bases (there can be more values separated by the space), where * stands for the name of the original column which is disambiguated. Together with Annotations (JSON) file it conforms to the [CSV on the Web](#) standard.

Example

```
District;           City;           POP_FOR_NAT;     POP_TOTAL;
City_url;  District_url;
Graz-Stadt;           Graz;           46952;
269997;           ...;           ...;
Deutschlandsberg;   Aibl;           39;           1386;
...;           ...;
```

Export process

In the export process all chosen cell annotations from the Result are read. Annotation for primary knowledge base is written to the *_url column (newly created, * stands for name of the corresponding original column), other annotations are written to the *_alternative_urls column. If the annotation for primary knowledge base is empty, one of other annotations is moved to the *_url column.

18 Statistical data specifics

When statistical data are processed, one more column (apart from columns with disambiguated URLs) is added to the extended CSV file. Its header name is "OBSERVATION_url" and values in cells are generated according to this template:

"{kbInsertDataElementPrefix}observation/{UUID}", where kbInsertDataElementPrefix is fetched from the KnowledgeBase configuration (property called kb.insert.prefix.data.element) and UUID is generated randomly. These URLs then serve as subjects in triples describing particular observations (every row is one observation).

18.1.1.1 RDF (triples)

RDF data are generated from the Annotated table (JSON) and extended Input (CSV) according to [csv2rdf](#) standard. We can export data in various RDF formats, for example Turtle or JSON-LD.

Example

```
<http://adequate.at/concept/city/aibl> a
<http://adequate.at/concept/city> ;
  dct:terms:title "City", "LAU_NAME", "Town" ;
  dc:description "City of Austria" ;
  x:hasPopulationNat
"39"^^<http://www.w3.org/2001/XMLSchema#integer> ;
  x:hasPopulationTotal
"1386"^^<http://www.w3.org/2001/XMLSchema#integer> ;
  ad:liesIn
<http://adequate.at/concept/district/deutschlandsberg> .
<http://adequate.at/concept/district/deutschlandsberg> a
<http://adequate.at/concept/district>
...

```

Export process

RDF export consists of two parts:

- fetch the triple patterns from Annotated table
- create the triples from Input rows according to the patterns

19 Fetch the triple patterns

We read all columns from the Annotated table and try to recognize triple patterns they describe. When the column has the `suppressOutput` boolean flag set to true, we do not consider this column (it just holds the URIs and does not describe triples). If the `propertyUrl` is empty, no triple can be created (the predicate is missing). The same situation applies for empty `aboutUrl` (missing subject). When `aboutUrl` is not a column link (enclosed in curly braces), the column describes just one fixed triple statement, so we add this statement to the RDF Model directly (it is used for the statistical data cube definition). Then we finally can create the triple pattern. When the `valueUrl` is empty, we create a `DataPropertyTriplePattern`, where the object value in the column is literal). Otherwise we have two more options: When the separator is empty, we create a `ObjectPropertyTriplePattern`, where subject pattern is link to some column and object pattern can also be the link to the column or literal, so we save also the `dataType`. Otherwise the separator exists and we create a `ObjectListPropertyTriplePattern`, where the linked object column contains more resources separated by given separator.

20 Create the triples

Then we iterate over set of rows from the extended Input and for each row try to create the RDF triple from each triple pattern. First we create the subject. We expect only link to the

column (in curly braces), So we find the value in given column and current row and when the value is not empty (and the column itself is not missing), we can create the IRI for subject.

IRI for predicate is already contained in the pattern, because we created it during creation of the pattern.

When the pattern is instance of `DataPropertyTriplePattern`, we assume that the object is literal, so when the value in given column and current row is not empty (and column is not missing), we can create literal for object (with `dataType xsd:string`). When there is instance of `ObjectListPropertyTriplePattern`, we must split the value according to given separator and for each resulting value create the IRI of object (so that more triples will be created). Finally it could be instance of `ObjectPropertyTriplePattern`, then the value can be IRI or literal. We can recognize it by checking the validity of IRI (valid IRI contains a colon). If it is literal and `dataType` is set and valid, we also set the `dataType` to the literal.

When we have the subject, predicate and objects created, we can add the new statements to the RDF Model. We use [RDF4J](#) library for creating the Model, which can be then easily exported by [Rio](#) to the specified RDF format.

21 Statistical data specifics

RDF export is processed the same way as for other than statistical data, it just includes processing of certain additional triples (without links to columns) and differentiates between creating of IRIs and Literals in object part of triples. Prefixes read from Annotated table "context" attribute are set to the RDF Model, so they are written to the header part of the resulting Turtle file.

21.1.1 Data cube

Statistical data can be published as RDF Data cube. General documentation of RDF Data cube vocabulary can be found at <http://www.w3.org/TR/vocab-data-cube/>. The following content of this section discusses information and proposals for RDF Data cube export functionality of Odalic and corresponding issues and problems that were encountered.

21.1.1.1 Input file structure

First we had to decide which structure of input file would be supported by Odalic for processing statistical data. There is an example of input file in documentation on the page <http://www.w3.org/TR/vocab-data-cube/#example>:

	2004-2006		2005-2007		2006-2008	
	Male	Female	Male	Female	Male	Female
Newport	76.7	80.7	77.1	80.9	77.0	81.5

Cardiff	78.7	83.3	78.6	83.7	78.7	83.4
Monmouthshire	76.6	81.3	76.5	81.5	76.6	81.7
Merthyr Tydfil	75.5	79.1	75.5	79.4	74.9	79.6

There are three dimensions: time period (rolling averages over three year time-spans), region and sex. Each observation represents the life expectancy for that population (the measure) and we needed an attribute to define the units (years) of the measured values. This table has multiline headers, heading rows and heading column. Then every cell represents one observation. But this structure of the table is not in the end supported by Odalic. Odalic supports only tables with exactly one header row and no header columns. So the data above can be transformed to following table structure (slightly extended):

Country	Region	Time period	Sex	Life expectancy
Count1	Newport	2004-2006	Male	76.7
Count1	Newport	2004-2006	Female	80.7
Count1	Newport	2005-2007	Male	77.1
Count1	Newport	2005-2007	Female	80.9
Count1	Newport	2006-2008	Male	77.0
Count1	Newport	2006-2008	Female	81.5
Count1	Cardiff	2004-2006	Male	78.7
Count1	Cardiff	2004-2006	Female	83.3
Count1	Cardiff	2005-2007	Male	78.6
Count1	Cardiff	2005-2007	Female	83.7
Count1	Cardiff	2006-2008	Male	78.7
Count1	Cardiff	2006-2008	Female	83.4
Count2	Monmouthshire	2004-2006	Male	76.6
Count2	Monmouthshire	2004-2006	Female	81.3
Count2	Monmouthshire	2005-2007	Male	76.5
Count2	Monmouthshire	2005-2007	Female	81.5
Count2	Monmouthshire	2006-2008	Male	76.6
Count2	Monmouthshire	2006-2008	Female	81.7
Count2	Merthyr Tydfil	2004-2006	Male	75.5

Count2	Merthyr Tydfil	2004-2006	Female	79.1
Count2	Merthyr Tydfil	2005-2007	Male	75.5
Count2	Merthyr Tydfil	2005-2007	Female	79.4
Count2	Merthyr Tydfil	2006-2008	Male	74.9
Count2	Merthyr Tydfil	2006-2008	Female	79.6

Then every row represents one observation. One column represents measure (Life expectancy) and three columns represent dimensions (Region, Time period, Sex). First column is neither measure nor dimension, because there is a relation between Country and Region. There are no relations among other columns. Theoretically there could be more columns representing measures.

21.1.1.2 Resulting RDF Data cube and the generated patterns

Based on the example above, there is complete resulting RDF Data cube in documentation at <http://www.w3.org/TR/vocab-data-cube/#full-example>. According to the example in documentation the RDF Data cube contains these parts:

- Data Set
- Data structure definition
- Dimensions and measures
- Observations

For every part there is a "pattern" showing how Odalic producec the RDF. For producing the RDF Data cube we needed the Result provided by Odalic core algorithm and also the Data cube definition ("CubeDef") provided by user. For every pattern there is depicted what information we need from Result and CubeDef for producing the RDF.

Data Set pattern

Input from Odalic Result:

- (none)

Input from user's CubeDef:

Parameter	Value
Title	Life expectancy title
Label	Life expectancy desc
Comment	Life expectancy within Welsh Unitary authorities comment

Description	Life expectancy within Welsh Unitary authorities - extracted from Stats Wales
Subject	http://purl.org/linked-data/sdmx/2009/subject/3.2
Organization	Example org

- Note: Date for "issued" can be computed by program during RDF producing.

RDF output pattern:

```
# -- Data Set -----
eg:dataset a qb:DataSet;
  dct:title      "Life expectancy title";
  rdfs:label     "Life expectancy desc";
  rdfs:comment   "Life expectancy within Welsh Unitary
authorities comment";
  dct:description "Life expectancy within Welsh Unitary
authorities - extracted from Stats Wales";
  dct:publisher  eg:organization;
  dct:issued    "2016-09-22";
  dct:subject    <http://purl.org/linked-
data/sdmx/2009/subject/3.2>;
  qb:structure   eg:dsd;
  .

eg:organization a org:Organization, foaf:Agent;
  rdfs:label "Example org";
  .
```

Data structure definition pattern

Input from Odalic Result:

- (none)

Input from user's CubeDef:

- Which columns are dimensions and measures - the column numbers (order in the Input of the task) are enough.

RDF output pattern:

```
# -- Data structure definition -----
eg:dsd a qb:DataStructureDefinition;
  # The dimensions
  qb:component [ qb:dimension eg:refArea;          qb:order 1 ];
  qb:component [ qb:dimension eg:refPeriod;        qb:order 2 ];
```

```

# The measure(s)
qb:component [ qb:measure eg:lifeExpectancy ];

# The attributes
qb:component [ qb:attribute sdmx-attribute:unitMeasure ];
.

```

Dimensions and measures pattern

Input from Odalic Result:

- Classification of columns pointed by user as dimensions and measures (Label and Resource)

Input from user's CubeDef:

- Which columns are dimensions and measures (for example the number of column in input is enough).

RDF output pattern:

```

# -- Dimensions and measures -----
eg:refPeriod a rdf:Property, qb:DimensionProperty;
  rdfs:label "reference period";
  qb:concept <http://dbpedia.org/resource/Reference_period>;
.

eg:refArea a rdf:Property, qb:DimensionProperty;
  rdfs:label "reference area";
  qb:concept <http://dbpedia.org/resource/Region>;
.

eg:lifeExpectancy a rdf:Property, qb:MeasureProperty;
  rdfs:label "life expectancy";
  rdfs:subPropertyOf sdmx-measure:obsValue;
  qb:concept <http://dbpedia.org/resource/Life_expectancy>;
.

```

Observations pattern

Input from Odalic Result:

- Disambiguation of cells in columns pointed by user as dimensions (Resource)

Input from user's CubeDef:

- Unit of measure (Resource)

Note: Values of cells in column pointed by user as measure are obtained from the Input of the task.

RDF output pattern:

```
# -- Observations -----  
  
eg:o1 a qb:Observation;  
  qb:dataSet  eg:dataset ;  
  eg:refArea  
<http://dbpedia.org/page/Newport,_New_South_Wales> ;  
  eg:refPeriod  
<http://reference.data.gov.uk/id/gregorian-interval/2004-01-  
01T00:00:00/P3Y> ;  
  sdmx-attribute:unitMeasure <http://dbpedia.org/resource/Year> ;  
  eg:lifeExpectancy          76.7 ;  
  .  
  
eg:o2 a qb:Observation;  
  qb:dataSet  eg:dataset ;  
  eg:refArea  
<http://dbpedia.org/resource/Cardiff> ;  
  eg:refPeriod  
<http://reference.data.gov.uk/id/gregorian-interval/2004-01-  
01T00:00:00/P3Y> ;  
  sdmx-attribute:unitMeasure <http://dbpedia.org/resource/Year> ;  
  eg:lifeExpectancy          78.7 ;  
  .
```

21.1.2 Configurations export and import

21.1.2.1 Motivation

The result of the execution is determined by the following factors:

- content of the processed file
- parsing format assigned to the file
 - e.g. changing the used delimiter can affect how many columns the CSV file has
- available knowledge bases

- e.g. when some resource is missing in the base, the algorithm will not use it to annotate any of the table parts
- task configuration, which stands for:
 - task description
 - the input file
 - provided feedback which the server uses as constraints for the next algorithm run
 - set of of bases, that the user selected to run the processing against
 - chosen primary base
 - specified maximum number of rows that will be processed from the file
 - whether to approach the input as statistical data (which ultimately results in the export of the [RDF data cude](#))

It is impractical to transfer the whole knowledge bases, the more so because they are usually remotely accessible. The definition of proxies on the other hand are small and easy to transfer. Files are also easy to send from one machine to another, the remote file location can just be shared (apart from the parsing format, but it hardly ever changes so it does not hurt to set it up on another machine). What remains to solve is the task configuration export and import.

In the current state the task configuration includes also the definitions of the used base proxies. When a base proxy of the same name is already present, it is used as it is; if not the serialized definition is used to create the used base proxy first. The base proxies can also be exported and imported independently on the tasks.

Tasks do also have other properties: owning user, task ID, time of creation/modification. But these are ephemeral and not that useful to keep in a configuration transferable from machine to machine.

21.1.2.2 Format choice

One has almost too many choices when choosing how to encode the configurations. Serialized RDF is an appealing option, not only because of the project subject matter, but it also makes it easy to accompany the processing results with their provenance. What makes the conversion to RDF a challenge is a relatively large and diverse class hierarchy of the task configuration that must be turned into RDF statements. It would be possible to write a code doing that by manually constructing the RDF model, but an option to annotate the involved classes and employ some framework to construct the statements automatically (as is the case when mapping the domain classes to JSON through JAXB annotations for the REST API) appeared like a better choice.

There are few options, such as [Alibaba](#), [Empire](#), but these are focused on storing Java objects in RDF stores and are therefore too heavy for simple round-trip conversion. There is an

existing older library <http://rdfbeans.sourceforge.net/>, which appears to be abandoned. Ultimately [Pinto](#) library was chosen. Following [the practice established to convert domain objects to JSON for the REST API](#), a separate package `cz.cuni.mff.xrg.odalic.api.rdf.values` containing the mapped versions of objects was established and these versions annotated with Pinto annotations. Because the Pinto lacks the concept of [XMLJavaTypeAdapters](#), the mapped version of objects must refer to other mapped versions, and methods converting the value objects back to the domain ones have to be provided.

Much larger complication appeared when attempting to convert Java Maps of more complex types than Strings. Pinto did not handle these cases well, so the maps had to be converted to collections of key-value entries first. Apart from that, the solution finalized in `cz.cuni.mff.xrg.odalic.api.rdf.TurtleRdfMappingTaskSerializationService` and in `cz.cuni.mff.xrg.odalic.api.rdf.TurtleRdfMappingknowledgeBaseSerializationService` proved to be reliable, even for a complicated configuration cases, involving extensive user feedback.

21.1.2.3 Format specification

All the exported tasks have a unique identifier generated, which is present in the serialized configuration in a triple in the form:

```
<http://odalic.eu/odalic/SerializedTask/V5/246c095d-f89b-4151-962b-34bd25b02843> a <http://odalic.eu/internal/Task>
```

The subject URI consists of three main parts: application instance web address (`odalic.eu` in this case), version identifier (currently the fifth version) and UUID. The subject is of type <http://odali.eu/internal/Task>, and through other RDF statements has all the other exported properties linked. Properties such as <http://odalic.eu/internal/Task/configuration> follow a common pattern where every property has a suffix in the form `http://odalic.eu/internal/`, followed by the name of the class the property belongs to and the name of the property, derived from the properties as defined by the objects exchanged through [REST API](#).

Knowledge base proxies follow the same schema, only substituting `Task` for `KnowledgeBase`. The underlying library does not map Java collections using RDF collections, but instead opts for one-time (but not anonymous) nodes forming the defining connections. They share the same prefix `http://odalic.eu/odalic/SerializedTask/Node/` followed by UUID. These are also used to represent contained entities (which alleviates the need to create manually unique identifier for each "pointer"). The only exception are maps which are before mapping converted to a set of entries (for example in the case of a map from base name to annotation candidates).

The typical fragment of exported configuration looks like this, which illustrates the above mentioned peculiarities of the format:

```

<http://odalic.eu/odalic/SerializedTask/V5/29eac34b-de44-4328-8d54-
ad97799384f7> a <http://odalic.eu/internal/Task> ;
  <http://odalic.eu/internal/Task/configuration>
<http://odalic.eu/odalic/SerializedTask/Node/8a0734da21ccdc62ad71f2
483c8519d8> .

<http://odalic.eu/odalic/SerializedTask/Node/f1188357b6a2c82746d36a
f42dd04cb3> a <http://odalic.eu/internal/Entity> .

<http://odalic.eu/odalic/SerializedTask/Node/6ea67dc770af060bc491db
b8385bc867> <http://odalic.eu/internal/Entity/resource>
"http://dbpedia.org/dbtax/Surname" .

<http://odalic.eu/odalic/SerializedTask/Node/6532423b372f99780ed96f
118409c1d1> <http://odalic.eu/internal/EntityCandidate/score>
<http://odalic.eu/odalic/SerializedTask/Node/928ec89c7f9e3d7a72590c
219e1cb611> .

<http://odalic.eu/odalic/SerializedTask/Node/f501ceaf6326453502a757
e2a22ddcfa> <http://odalic.eu/internal/KnowledgeBase/insertGraph>
"http://odalic.eu" .

<http://odalic.eu/odalic/SerializedTask/Node/59d958855ae15086db58fe
a03a825b6f> a <http://odalic.eu/internal/Entity> .

<http://odalic.eu/odalic/SerializedTask/Node/4b072229a26e9e103703e0
60b1133020> <http://odalic.eu/internal/EntityCandidate/entity>
<http://odalic.eu/odalic/SerializedTask/Node/156ae04d9c18edca709e51
a644076014> .

<http://odalic.eu/odalic/SerializedTask/Node/156ae04d9c18edca709e51
a644076014> <http://odalic.eu/internal/Entity/label> "Person" .

<http://odalic.eu/odalic/SerializedTask/Node/fdcd19e5f18a7469262778
efed4bc633>
<http://odalic.eu/internal/EntityCandidateNavigableSetWrapper/value
>
<http://odalic.eu/odalic/SerializedTask/Node/239164724ee31eea35337b
2d0a83a21b> .

<http://odalic.eu/odalic/SerializedTask/Node/75734a569feba5ad18339a
2dbfa7cab7>
<http://odalic.eu/internal/KnowledgeBaseEntityCandidateNavigableSet
Entry/base> "DBpedia" .

<http://odalic.eu/odalic/SerializedTask/Node/6888029707e6588411dbbe
33f195597a> <http://odalic.eu/odalic/SerializedTask/Node/value>
3.0438887148351625E0 .

```


...

21.1.3 Persisting server state

21.1.3.1 Motivation

Extraction of the Linked Open Data is an expensive process. The knowledge bases are usually accessed remotely, and even if they are mirrored locally or most of the requests hit the cache, the processing can take from several seconds up to several hours. Therefore it is crucial that once computed results are not involuntarily lost, and there is no need to run it again. Apart from that the requirements on the permanent storage by the Odalic Semantic Table Interpretation are relatively modest. There is only a handful of classes which instances has to be serialized:

- Users attempting registration.
 - Otherwise the sign up requests would get lost and confirmation links sent in e-mails would not work.
- Registered users.
 - This requirement is natural.
- Logged in users.
 - This could be eventually omitted, but since the tokens distributed by the application might be relatively long-lived (as is the case of the UnifiedViews plugin), it is better to keep it safe.
- Files data and meta-data.
 - Even in the case of remote files, there is strong need to keep the format used by the parser intact.
- Files utilization by the tasks.
 - Files can be shared between several task, if we loose this information, a file might accidentally get deleted, despite being a dependency of some configured task.
- Task configurations.
 - This requirement could be theoretically partially solved by exporting the definitions and re-importing them in time of need. But this is impractical in large numbers. Also the configurations do not keep references to the owning users (to allow easy exchange and archiving) and the files are linked only by their ID.
- Results.

- As mentioned in the introducing statement, all the tasks could be potentially re-run, but there are good reasons not to.

21.1.3.2 MapDB

Embedded database [MapDB](#) was selected as the solution for keeping the state from several reasons:

- There is no need to somehow convert or map the stored objects. They only need to be serializable and immutable, which is a good idea anyway.
- Its interfaces are common Java collections and maps, therefore its usage is almost seamless in simple cases.
 - This came handy because all of the aforementioned services were written as memory-only first (and still can be set as, through the Spring configuration), where the objects were kept in the Java collections and maps.
- It comes with [write-ahead-log and transactions](#).
- It is performing [well enough](#).
- It has a decent [support for nested maps](#), called prefix tables, similar to [Guava Tables](#).
 - This served well when the time to extend the existing code to provide separate user spaces came. It practically meant only to introduce a top-level map from user IDs to the previously used maps.

Despite being in version 3, it still has some minor flaws, which were not an obstacle to its deployment in Odalic:

- The transactions are limited only to collection instances spawned from the same DB object. This makes them hard to execute across method boundaries without sharing the DB project. This is a major design nuisance, which one does not usually encounter when relying on mainstream Java Transaction API.
- Also all commits and roll-backs has to be explicit.
- In comparison to previous version, the 3rd one is forcing its users to cast map keys to Objects in order to use the prefix tables. This makes the code inherently less safe (and indeed we encountered a bug that was hidden behind this, when testing the Odalic server).

Usage within the project

All stored objects are kept in collections and maps initialized from a single MapDB object, which is obtainable by calling a method `getDb()` at `cz.cuni.mff.xrg.odalic.util.storage.FileDbService`, which implements interface

cz.cuni.mff.xrg.odalic.util.storage.DbService. User can specify the location of the used writeable file at key cz.cuni.mff.xrg.odalic.db.file in the config/sti.properties.

For example initialization of a map from user IDs to the user instances looks like this:

```
this.db = dbService.getDb(); // Kept for further use, mainly
committing of transactions.
this.userIdsToUsers = this.db.hashMap("userIdsToUsers",
Serializer.STRING, Serializer.JAVA).createOrOpen(); // Uses a more
performant MapDB serializer for the keys.
```

Typical write to the DB can be found in a method

cz.cuni.mff.xrg.odalic.users.DbUserService.confirmPasswordChange(Token), which demonstrates how the transactions work:

```
@Override
public void confirmPasswordChange(final Token token) {
    final DecodedToken decodedToken = validateAndDecode(token);

    final User newUser = matchPasswordChangingUser(decodedToken);

    final User replaced;
    try {
        replaced = replace(newUser);
    } catch (final Exception e) {
        this.db.rollback(); // Rollback in case the user version
replacement fails.
        throw e;
    }

    invalidateTokens(replaced);

    this.db.commit(); // Commit in case of success.
}

private User replace(final User user) {
    final User replaced =
this.userIdsToUsers.replace(user.getEmail(), user);
    Preconditions.checkNotNull(replaced, "Nonexisting user!");

    return replaced;
}
```

21.1.4 Configuration

The Odalic server has one main configuration file with general settings and several specialized configuration files for various parts of the server. All of the specialized configuration files are referenced in the main file either directly, or through other specialized files. The path to the main configuration file is set through the environment variable **cz.cuni.mff.xrg.odalic.sti**.

Code Block 9 Example VM options:

```
-Dcz.cuni.mff.xrg.odalic.sti=c:\odalic\sti\config\sti.properties
```

[Main Settings](#)

[KB Proxy Settings](#)

[KB Structure Settings \(Groups\)](#)

[Websearch Settings](#)

21.1.4.1 Main Settings

All main settings are **mandatory**, unless it is stated otherwise in the description.

File Paths

- **sti.home** - STI home folder. All other file paths will be relative to this (except otherwise stated).

Code Block 10 Example

```
sti.home = d:\\Documents\\odalic\\sti\\
```

- **sti.nlp** - Folder containing nlp resources, by default this is the "/resource" folder in the distribution. The resources generally contain data that may change during the processing, so it makes sense to keep them apart from the configuration.

Code Block 11 Example

```
sti.nlp = resources
```

- **sti.cache.main.dir** - Folder containing cached data. Whenever KB search or Web search is performed, the query and results are cached in a Solr instance. This specifies the base path of all Solr instances.

Code Block 12 Example

```
sti.cache.main.dir = ..\\cache\\
```

- **sti.websearch.properties** - Web search configuration files. By default, it is in the "/config" folder of the distribution. See [Websearch Settings](#).

Code Block 13 Example

```
sti.websearch.properties = config\\websearch.properties
```

- Former **sti.kbproxy.propertyfile** option, which contained a list of bases to load, has been made obsolete thanks to introduction of runtime KB management. Nevertheless the user can still leave KB proxy configuration files in the config library and they are scanned during startup to create the initial set of available bases for a newly signed-up user. This helps to provide examples to customize, while keeping the old files format which is arguably easier to manually customize than exported RDF configurations. These should be put in the "/config" folder of the distribution. See [KB Proxy Settings](#) for details of the files.

Subject column detection, the ws scorer

- **sti.subjectcolumnndetection.ws** - Choose whether STI should use the web search score in detecting subject columns (the columns serving as the source of relations with other columns).

Code Block 14 Example

```
sti.subjectcolumnndetection.ws = true
```

- **sti.iinf.websearch.stopping.class** - If the above mentioned option is on, determines what stopping criteria class should be used. Must extend the **uk.ac.shef.dcs.sti.core.algorithm.tmp.stopping** class.

Code Block 15 Example

```
sti.iinf.websearch.stopping.class =  
uk.ac.shef.dcs.sti.core.algorithm.tmp.stopping.IInf
```

- **sti.iinf.websearch.stopping.class.constructor.params** - If the ws option is on, this provides the stopping criteria class above its constructor parameters in the order as defined, delimited by ",". These options generally serve to fine-tune the algorithm results and are hardly ever needed to be modified.

Code Block 16 Example

```
sti.iinf.websearch.stopping.class.constructor.params =  
0.0,1,0.01
```

Relation enumeration

- **sti.learning.relation** - Choose whether or not STI should annotate relations. May be turned off when this is not required, for example when the administrator wants to rely only on the relations between columns and not put in relation the individual row cells in the results.

Code Block 17 Example

```
sti.learning.relation = true
```

Output

- **cz.cuni.mff.xrg.odalic.prefixes** - Prefix mapping configuration for the resources provided to the clients. Defines commonly used URI prefixes. A good candidate to turn into runtime option in later releases.

Code Block 18 Example

```
cz.cuni.mff.xrg.odalic.prefixes = config\\PrefixMapping.ttl
```

Tableminer+

- **sti.tmp.iinf.learning.stopping.class** - What stopping criteria class should be used in the iinf for preliminary column classification. Must extend the **uk.ac.shef.dcs.sti.core.algorithm.tmp.stopping** class.

Code Block 19 Example

```
sti.tmp.iinf.learning.stopping.class =  
uk.ac.shef.dcs.sti.core.algorithm.tmp.stopping.IInf
```

- **sti.tmp.iinf.learning.stopping.class.constructor.params** - For the stopping criteria class provided above, also provide its constructor parameters in the order it is defined in the class values separated by ",".

Code Block 20 Example

```
sti.tmp.iinf.learning.stopping.class.constructor.params =  
0.0,1,0.05
```

Mail (for confirmation emails)

- **mail.username** - SMTP server user name.
- **mail.password** - SMTP server password.
- **mail.from** - Sender of the outgoing e-mails.
- **mail.smtp.host** - Address of the SMTP server.
- **mail.smtp.auth** - Choose whether or the SMTP server use authentication.
- **mail.smtp.port** - Port of the SMTP server.
- **mail.smtp.socketFactory.class** - Class used as socket factory.
- **mail.smtp.socketFactory.port** - Port used by the socket factory.

Code Block 21 Example

```
mail.username = odalic@email.cz  
mail.password = password  
mail.from = odalic@email.cz  
mail.smtp.host = smtp.seznam.cz  
mail.smtp.auth = true  
mail.smtp.port = 465  
mail.smtp.socketFactory.class = javax.net.ssl.SSLSocketFactory  
mail.smtp.socketFactory.port = 465
```

Users

- **cz.cuni.mff.xrg.odalic.users.maximumCodesKept** - Maximum temporary codes kept per queue. When the confirmation of sign-up and password change is on, this helps to prevent to grow the waiting queue out of proportions.
- **cz.cuni.mff.xrg.odalic.users.session.maximum.hours** - Maximum hours that a single session can last.
- **cz.cuni.mff.xrg.odalic.users.signup.window.minutes** - Length of the time window when the sign-up confirmation token is active (in minutes).
- **cz.cuni.mff.xrg.odalic.users.reset.window.minutes** - Length of the time window when the password setting confirmation token is active (in minutes).

- **cz.cuni.mff.xrg.odalic.users.signup.url** - URL format for a sing-up confirmation. This is put in the confirmation e-mails in order the lead the user to the correct running UI instance.
- **cz.cuni.mff.xrg.odalic.users.reset.url** - URL format for a password setting confirmation. This is put in the confirmation e-mails in order the lead the user to the correct running UI instance.
- **cz.cuni.mff.xrg.odalic.users.admin.email** - Administrator's email (and the user ID).
- **cz.cuni.mff.xrg.odalic.users.admin.password** - Administrator's initial password. It is recommended to change it using the standard REST API or UI means.

Code Block 22 Example

```

cz.cuni.mff.xrg.odalic.users.maximumCodesKept = 100
cz.cuni.mff.xrg.odalic.users.session.maximum.hours = 172
cz.cuni.mff.xrg.odalic.users.signup.window.minutes = 30
cz.cuni.mff.xrg.odalic.users.reset.window.minutes = 30
cz.cuni.mff.xrg.odalic.users.signup.url =
http://localhost:8080/odalic-ui/index.html#/signup/%s
cz.cuni.mff.xrg.odalic.users.reset.url =
http://localhost:8080/odalic-ui/index.html#/chnpasswd/%s
cz.cuni.mff.xrg.odalic.users.admin.email = odalic@email.cz
cz.cuni.mff.xrg.odalic.users.admin.password = admin

```

Tokens

- **cz.cuni.mff.xrg.odalic.tokens.issuer** - Name of the organization issuing this instance tokens. Serves mainly to distinguish tokens from non-related instances of the application.

Code Block 23 Example

```

cz.cuni.mff.xrg.odalic.tokens.issuer = Odalic

```

- **cz.cuni.mff.xrg.odalic.tokens.secret** - Secret used for generating and verifying authentication tokens.

Code Block 24 Example

```

cz.cuni.mff.xrg.odalic.tokens.secret =
cPLsKpTZxAcDZH5cqq3bxAet3VdAJ683X8Ccu8yTyFh

```

Storage

- **cz.cuni.mff.xrg.odalic.db.file** - The local database file used to store Odalic data.

Code Block 25 Example

```
cz.cuni.mff.xrg.odalic.db.file = resources\\db.dat
```

21.1.4.2 KB Proxy Settings

Common Settings

Mandatory common settings:

- Former option **kb.cacheTemplatePath** has been eliminated, the caches are dynamically created in the main cache folder for each user and his or her base.
- **kb.advancedType**- The name of the type. May affect interpretation of advanced key-values defined for the Knowledge base proxy.

Code Block 26 Example

```
kb.advancedType = SPARQL
```

- **kb.name** - The name of the knowledge base. Should be unique.

Code Block 27 Example

```
kb.name = DBpedia
```

- **kb.stopListFile** - A file that lists things (e.g., predicate URIs) to be used by an instance of **uk.ac.shef.dcs.kbproxy.KBSearchResultFilter**, which decides what triples/reasons/classes to remove from the result as they are too general or meaningless. Specific rules are implemented by subclasses of **uk.ac.shef.dcs.kbproxy.KBSearchResultFilter**, which should be instantiated as part of **uk.ac.shef.dcs.kbproxy.KBProxy**.

Code Block 28 Example

```
kb.stopListFile = resources\\kbstoplist_dbpedia.txt
```

- **kb.endpoint** - SPARQL endpoint of the knowledge base.

Code Block 29 Example

```
kb.endpoint = http://localhost:8896/sparql
```

- **kb.structure.groups** - The list of predicates used in the knowledge base. Multiple configuration files can be separated by the "|". See [KB Structure Settings \(Groups\)](#).

Code Block 30 Example

```
kb.structure = dbpedia|rdf|skos
```

Insert Settings

- **kb.insert.supported** - Enables inserting new concepts into the knowledge base.

```
kb.insert.supported = false
```

Mandatory insert settings when **kb.insert.supported** is true

- **kb.insert.prefix.data** - Prefix used in data elements (instances of classes).

Code Block 31 Example

```
kb.insert.prefix.data = http://odalic.eu/resource/
```

- **kb.insert.prefix.schema** - Prefix used in schema elements (classes and properties).

```
kb.insert.prefix.schema = http://odalic.eu/schema/
```

- **kb.insert.graph** - Named graph used for new concepts.

Code Block 32 Example

```
kb.insert.graph = http://odalic.eu
```

- **kb.fulltextEnabled** - Enables full-text search in the knowledge base. If a query keyword does not match to anything, it tries to split it into parts using "and".

Code Block 33 Default

```
kb.fulltextEnabled = true
```

- **kb.useBifContains** - Enables support for the Virtuoso full-text search. Does nothing if the full-text search is disabled.

Code Block 34 Default

```
kb.useBifContains = true
```

- **kb.languageSuffix** - The language suffix used in exact string matching and labels of newly added concepts.

Code Block 35 Default

```
kb.languageSuffix =
```

Optional insert settings (the implementation is free to ignore and even when omitted, the shown values are used as defaults).

- **kb.insert.endpoint** - When present it is used to insert resources instead of the default one. Some bases do have separate endpoint for insertion.

```
kb.insert.endpoint =
```

- **kb.insert.defaultClass** - Default class used in situations, when the class is not specified.

Code Block 36 Default

```
kb.insert.defaultClass = http://www.w3.org/2002/07/owl#Thing
```

- **kb.insert.predicate.label** - Predicate used for assigning labels.

Code Block 37 Default

```
kb.insert.label = http://www.w3.org/2000/01/rdf-schema#label
```

- **kb.insert.predicate.alternativeLabel** - Predicate used for assigning alternative labels.

Code Block 38 Default

```
kb.insert.label = http://www.w3.org/2000/01/rdf-schema#label
```

- **kb.insert.predicate.subclassOf** - Predicate used for the "subclass of" relationship.

Code Block 39 Default

```
kb.insert.predicate.subclassOf =  
http://www.w3.org/2000/01/rdf-schema#subClassOf
```

- **kb.insert.predicate.subPropertyOf** - Predicate used for the "sub-property of" relationship.

Code Block 40 Default

```
kb.insert.predicate.subProperty =  
http://www.w3.org/2000/01/rdf-schema#subPropertyOf
```

- **kb.insert.type.class** - Type used for inserting classes.

Code Block 41 Default

```
kb.insert.type.class = http://www.w3.org/2002/07/owl#Class
```

- **kb.insert.type.property** - Type used for inserting properties.

Code Block 42 Default

```
kb.insert.type.property = http://www.w3.org/1999/02/22-rdf-  
syntax-ns#Property
```

- **kb.insert.type.dataProperty** - datatype properties type

```
kb.insert.type.dataProperty =  
http://www.w3.org/2002/07/owl#DatatypeProperty
```

- **kb.insert.type.objectProperty** - object properties type

```
kb.insert.type.objectProperty =  
http://www.w3.org/2002/07/owl#ObjectProperty
```

Other **optional** settings (implementation is free to ignore them too).

- **kb.classTypeMode** - During initial disambiguation, class restriction is applied to the disambiguated entity. There are two modes, in which the class restriction may be applied. In the standard "indirect" mode, the disambiguated entity must be instance of something, which is a class. In the direct mode, the disambiguated entity can be class itself - this is the case e.g. in case of SKOS schemas, where there are just skos:Concepts.

Code Block 43 Default

```
kb.classTypeMode = indirect
```

- **kb.structure.predicate.instanceOf** - Predicate used for the "instance of" relationship.

Code Block 44 Default

```
kb.structure.predicate.instanceOf =  
http://www.w3.org/1999/02/22-rdf-syntax-ns#type
```

- **kb.structure.predicate.domain** - Predicate used to specify domains of properties.

Code Block 45 Default

```
kb.structure.predicate.domain = http://www.w3.org/2000/01/rdf-  
schema#domain
```

- **kb.structure.predicate.range** - Predicate used to specify ranges of properties.

Code Block 46 Default

```
kb.structure.predicate.range = http://www.w3.org/2000/01/rdf-  
schema#range
```

21.1.4.3 KB Structure Settings (Groups)

These settings define the structure of the knowledge base. Each setting can contain multiple values separated by the space character " ". Mandatory settings must be defined in at least one group per knowledge base proxy. For the initially loaded bases, these are located in the enums subdirectory. The base configurations can either specify them manually or let the application detect them manually before each task run when the key `kb.structure.groups` is omitted in the KB proxy definition.

Types

All settings in this category are **optional**.

- **kb.structure.type.class** - Types used for classes. This setting is by default undefined.

Code Block 47 Example

```
kb.structure.predicate.type = http://www.w3.org/1999/02/22-rdf-  
syntax-ns#type
```

- **kb.structure.type.property** - Types used for properties. This setting is by default undefined.

Code Block 48 Example

```
kb.structure.type.property = http://www.w3.org/1999/02/22-rdf-  
syntax-ns#Property
```

Predicates

Mandatory predicates (in the sense that at least one of the groups used by the must have them defined).

- **kb.structure.predicate.label** - Predicate used for assigning labels.

Code Block 49 Example

```
kb.structure.predicate.label = http://www.w3.org/2000/01/rdf-  
schema#label
```

- **kb.structure.predicate.type** - Predicate used for the "type" relationship.

Code Block 50 Example

```
kb.structure.predicate.type = http://www.w3.org/1999/02/22-rdf-  
syntax-ns#type
```

Optional predicates.

- **kb.structure.predicate.description** - Predicate used for assigning descriptions. This setting is by default undefined.

Code Block 51 Example

```
kb.structure.predicate.description =  
http://dbpedia.org/ontology/abstract
```

21.1.4.4 Websearch Settings

All websearch settings are **mandatory**.

The Websearch Implementation

- **web.search.class** - What class should be used to do web search (used in subject column detection for computing the ws score). It must extend `uk.ac.shef.dcs.websearch.WebSearch` and `uk.ac.shef.dcs.websearch.WebSearchFactory` must be revised to instantiate it by reflection.

Code Block 52 Example

```
web.search.class = uk.ac.shef.dcs.websearch.bing.v5.BingSearch
```

BingSearch specific settings

- **bing.key** - This is the API key to be used with `BingSearch`. You should apply for your own at: <https://datamarket.azure.com/dataset/bing/searchweb> (Bing Search API, Web Results Only).

Code Block 53 Example

```
bing.key = e413ab08c2e74283b12205f9453350ee
```

- **bing.url** - Default URL to access Bing Search API (Web Results Only).

Code Block 54 Example

```
bing.url =  
https://api.cognitive.microsoft.com/bing/v5.0/search?q=
```

21.1.5 RDF manipulation

The server has to deal in some places with the need to create, manipulate, import, export, query or otherwise interact with RDF data. There are already several established libraries or even frameworks that provide these services directly or became a part of other libraries, which in turn have become the Odalic dependencies. The most prominent ones are:

- [RDF4J](#)
 - While there are some dependencies (such as the Pinto library used to annotate classes for export and import, which did not move to new version from the 2.7 yet) that require to use its older versions simultaneously to the recent ones, there is certainly deficit in the fact the explicit usage (in the algorithm relatively old version or export preprocessing, fairly recent one) could be reduced to one version, thus benefiting from the reduced size of the deployed archive, faster start-up and easier future maintenance.
- [Jena](#)
 - While it may seem that many of the use cases are covered by RDF4J, Jena still found its use thanks to its capabilities in constructing of the SPARQL queries and potentially useful OWL support.

22 UI

22.1.1 Introduction

We decided to go with the approach of developing a single page application (SPA), which would communicate with the server via REST application interface. This way the server is lighter and does not have to deal with issues on the client's side. It allows for the server to be (almost) completely isolated, more secure, and puts weight of computing page look solely on the client's computer, which in turn increases the overall throughput of the server. Modern internet browsers make this approach possible on almost every popular platform.

We decided to stick with the basic personal computers as our platform of choice with installed browser(s) that support(s) HTML5, such as Microsoft[®] Edge 14 (*or newer*), Mozilla Firefox 50 (*or newer*) and Google[®] Chrome 49 (*or newer*). We used JavaScript language and Angular

(<https://angularjs.org/>) framework for the overall UI development, which would also determine the application architecture. Angular was chosen based on its popularity, good documentation and robustness, providing the desired functionality for almost everything we needed.

22.1.2 Architecture

Angular separates the entire application into several 'views'. It basically maps a route to a template which is then used for display. We usually call these 'views' as 'screens'. Each screen consists of a template, which is a portion of HTML code, and controller, being a piece of JavaScript code that 'controls' the template, providing functionalities for user interaction. During development, our goal was to put most of the code that determines visuals and contents of each screen into templates, while leaving controllers 'clean', providing only the necessary functions requested by templates. This also explains our certain design decisions, some of them will be described later.

Certain sub-parts of views naturally occur more than once in an application. For that reason Angular comes with a notion of directives: a pieces of code (sometimes) with their own templates attached. These may be later reused in views, or other directives. Last, but not least, for implementation of business logic, Angular provides services, which can be used by any controller, or another services.

22.1.3 Folder structure

The project is structured in the following way:

- **font:** external font files used by the application
- **graphics:** images used by the application
- **less:** stylesheets in the CSS/LESS format
- **css:** pre-compiled stylesheets from **less** folder
- **lib:** external libraries
- **test:** exemplar input files and responses from the server
- **src:** application source files
 - **common:** directives in the role of subcomponents, such as navigation bar, page footer, etc.
 - **directives:** generic reusable directives, such as modal window, pagination, tooltips, etc.
 - **filters:** generic custom angular filters

- **services:** implementation of business logic, such as REST services, authentication, etc.
- **templates:** application screens / views
 - may be structured to subcomponents
- **util:** generic utilities

index.html, contained within the root directory, is the application's entry point.

22.1.4 External libraries

As mentioned, for the overall development, Angular framework was used. Additional used libraries include:

- **bootstrap:** a popular, comprehensive CSS framework (<http://getbootstrap.com/>), speeding up overall development, providing basic styling and allowing for easy implementation of responsive pages
- **d3:** a library selected for implementation of graphvis component (*see Graphvis below*)
- **date:** provides basic functions for working with dates
- **fontawesome:** a popular, icon-based, font for improving overall appearance of the application
- **jquery:** better integration with bootstrap as well as speeding up development of directives
- **satellizer:** JWT based authentication / authorization library
- **uibootstrap:** a collection of several bootstrap modules, packed as directives, for better integration with Angular framework
- **less:** interpreter of CSS/LESS stylesheets, used only during development (*to optimize the application, the CSS/LESS stylesheets are pre-compiled*)
- **papaparse:** parser of CSV files; for testing purposes only, when running application without the server

22.1.5 Goals

As mentioned, during development our goal was to keep the application extensible (where possible) while also allowing for easy changes for the common cases. That is why we strived to keep logic and visuals in screens as much as possible separated. Most of the time controllers of screens do actually provide only the functionalities necessary, while leaving visuals solely up to templates of the screens. This may make certain development decisions look illogical to some as demonstrated by the following example, but helping us achieve our goal in the end:

Template:

```
...
<injector for="msgtxt.configdFailure">An error occurred while
trying to download the task's configuration.</injector>
...
```

Corresponding controller:

```
...
$scope.messages.push('error', $scope['msgtxt.configdFailure']);
...
```

The example relates to a situation when there is an uncaught error that needs to be displayed to a user. While it was possible to declare the message in the controller, by injecting it via template we can keep controllers 'clean' and allow for easy changes in the future.

22.1.6 Application loading

index.html being the entry point of the application references scripts used via `<script src="...">` HTML element. All external libraries are loaded this way and the main source file, **src/global.js**, defines the way of loading all other resources. It looks up **src/require.json** file for determining what components need to be loaded with an exception of screens, which are then loaded via **src/templates/templates.js**. Usage of JSON files, such as **src/require.json**, allows for easier extensibility of the application, requiring programmer only to write what additional components (s)he would like to be loaded, instead of editing the actual code. This adds to clarity and speeds up development. The components (*in this case we mean views, directives, services, ...*) then may be separated into several files and may consist of other subcomponents. However, loading of those has to be handled by the corresponding component itself.

Loading of screens is a little less straightforward. Screens have to be mapped to a certain route. Additionally it may be useful not to have a custom controller specified for certain screens, only a template, where there is no functionality necessary (*take for example home screen displaying basic information about the project, while providing no user interaction at all*). For clarity we decided to represent this in **src/templates/mappings.json**. Each screen is represented with an object:

```
{
  "route": "/signup/:token",
  "folder": "signupcnf",
  "controller": "odalic-signupcnf-ctrl"
```

```
}
```

while "route" being the route the screen is mapped to, "folder" being the name of a folder the screen is located in (*omitting **src/templates/***) and "controller" being a name of the corresponding controller. The name has to match the one specified in the actual controller definition. Additionally it may be equal to "generic" in which case it is assumed no custom controller is needed and only the template is loaded for the route. (*Note that in this case simply an empty controller is created automatically.*)

Please note the AngularJS supports a way to map a screen not only to a specific single route, but also to a *pattern*. In the case of `/signup/:token`, upon visiting `/signup/anything`, the mapping still holds. Not only that, we can also retrieve what `:token` part of the route is equal to (*in this case "anything"*) and specify further action based on this information. Such mechanism is used, for instance, by a sign up confirmation screen, which retrieves token directly from the route (*route being visited by a user upon receiving an e-mail to 'confirm the sign up by visiting the following link: <http://.../signup/GcOjUz1...>*), sends it to server for evaluation and displays information about the state of sign up process to the user.

For redirecting purposes there is another type of object that may be specified:

```
{
  "route": "/home",
  "target": "/",
  "controller": "reroute"
}
```

while "route" being the route the "redirect" is mapped to, "target" being the route to redirect to, and "controller" set to "reroute" (for clarity).

Name of the template has to always be **template.html** while controller has to be named **controller.js**. Loading of subcomponents / other files is handled individually.

22.1.7 Screens

The whole application is divided into screens. While some of their elements are the same (*e.g. navigation bar, footer, ...*), for the application to be as flexible as possible, none of the elements (*except header*) are hard-coded in the entry point. This allows for easier implementation of special cases, e.g. when the footer on a certain screen is to be different, or no present at all, etc.

In order to not repeat large amount of code throughout the screens, we separated common screen elements into 3 subcomponents:

- `main-cnt`: represents a wrapper around 'main content', i.e. the whole screen content needs to be contained within this element.
- `navbar`: a configurable navigation bar, has to be put inside `main-cnt` element.
- `footer`: a generic footer same for most of the screens.

That means an ordinary screen template will look like this:

```
<!-- Main Content -->
<main-cnt>
  <!-- Navigation Bar -->
  <navbar selected="home" lmenu="default-lm.json" rmenu="default-
rm.json"></navbar>

  <!-- Content + Sidebar -->
  <div class="container-fluid">
    <!-- Sidebar -->
    <div class="col-sm-3">
      Sidebar content
    </div>

    <!-- Content -->
    <div class="col-sm-9">
      Main content
    </div>
  </div>
</main-cnt>

<!-- Footer -->
<footer/>
```

22.1.7.1 Navbar

`navbar` is a configurable navigation bar, which means on each screen it can be set what items should be available and what item should be highlighted as the selected one. Available items have to be specified as a relative path to a file in JSON format. Specifically, there has to be a file describing menu on the left and a file describing menu on the right on the navigation bar. These two differ in how they work.

Menu on the left is an array of objects of the following format:

```
{
  "id": "home",
```

```
    "title": "Home",
    "link": "#/",
    "menu": []
  }
```

The exemplary object describes a single item on the navigation bar. "id" stands for identifier of the item. This can be referenced when describing what item is selected on a current screen. "title" stands for text displayed, "link" for where to redirect upon click and finally "menu" is an array of subitems. If the "menu" is not an empty array, "link" property should be omitted. The "menu" items look like this:

```
{
  "title": "File list",
  "link": "#/filelist"
}
```

where "title" stands for text displayed, "link" for where to redirect upon click. If no properties are provided, the item is identified as a separator (*visual purposes*).

Menu on the right consists of following type of objects:

```
{
  "id": "signup",
  "title": "Sign up",
  "link": "#/signup",
  "icon": "glyphicon-user",
  "condition": "!$auth.isAuthenticated()"
}
```

While being for the most part the same as objects in the left menu, there are some differences:

- Items on the right have an icon attached. See <http://getbootstrap.com/components/> reference for allowed icons.
- Items on the right may not represent a menu of subitems.
- Items on the right may specify a "condition" property, which determines whether should the item be displayed or not. Note that conditions are evaluated during each Angular digest cycle, therefore complicated conditions may cause performance issues.

22.1.8 Services

ODALIC being a single page application, most of the logic is handled on the server. Data exchanged between client and the server is realized via asynchronous requests (AJAX) and are (mostly) in JSON format. For data exchange a **"rest"** service (*actual name of the service*) is implemented, which transforms data sent/received, automatically injects headers required by the API and generally eases the overall work with the server's interface. The service is divided into parts corresponding to the ones described in [REST API specification](#).

To handle generic AJAX requests a **"requests"** service is implemented. Most of the responses from the server have a standardized format and therefore can be automatically parsed and transformed. The service handling this is injected via `"ioc['requests']"`, where **"ioc"** is a service being a very simple implementation of IoC (*inversion of control*) pattern (*src/services/ioc/modules.json* is the configuration file). The `"ioc['requests']"` additionally handles the case of unauthorized access to resources (*redirecting to log in screen by default*).

22.1.9 Authentication and authorization

As described in [Authentication and authorization](#), to ensure security, JWT (*JSON Web Tokens*) standard is used. For this we used Satellizer library (<https://github.com/sahat/satellizer>), which automatically signs each AJAX request with an appropriate authorization header and allows for easy token storage on a user's computer. Additionally, the library conveniently handles requests for logging and signing up.

Several screens are associated with the authentication / authorization process:

- **signup**: allows users to sign up
- **signupcnf**: maps to a route received by a user in an e-mail (*the e-mail requesting user's confirmation for signing up*); automatically handles additional requests associated with the process and displays notifications about the current state
- **login**: allows users to log into or out of the application, re-checks token (*to ensure its validity*) and displays the current state
- **chnpasswd**: allows users to change their password
- **chnpasswdcnf**: maps to a route received by a user in an e-mail (*the e-mail requesting user's confirmation for changing his/her password*); automatically handles additional requests associated with the process and displays notifications about the current state

22.1.10 Directives

While directives were designed to solve many different kinds of issues, there are some patterns we followed during development. We will demonstrate our approach, when designing directives, on an example of a confirmation modal window.

A confirmation modal window is a piece of HTML code consisting of several divisional elements (<div>) with correctly attached pre-defined classes. The classes as well as functionality is provided by the bootstrap framework. Therefore an example of how a modal window may look like, may be found here:

http://www.w3schools.com/bootstrap/bootstrap_modal.asp. What remains is a way to open the modal window and a way to close it from outside the directive. (*Note bootstrap already supports a way to open / close the modal window; however, a concrete element has to be selected first, which is rather impractical to do inside of a screen's controller.*) For that purpose we reserved a single attribute, "bind", acting as a 'gate' to our directive's interface. An example can be seen below:

Inside of a template we can use the directive the following way:

```
<!-- Confirm modal window -->
<confirm bind="myobj" title="Title">
  confirm modal window content, 'yes or no' question
</confirm>

<!-- Button to open the modal window with -->
<button ng-click="open()">Open</button>
```

Inside of a corresponding controller we may specify the following code:

```
// Initialization; the object will be filled by the corresponding
functions automatically
scope.myobj = {};

// On button click
scope.open = function() {
  // Open the modal
  scope.myobj.open(function (response) {
    // Upon closing the modal, this will be automatically
called
    if (!response) {
      console.log('A user answered "no".');
    }
  });
};
```

It is worth mentioning that we strived to put visually related interface into separate attributes (e.g. *modal headline is determined by the "title" attribute inside of a template*). "bind" attribute serves mostly as a 'gate' to call a directive's functions. For data or functions to be consumed by a directive itself we usually created individual attributes, as demonstrated by the following example:

```
<button-load button-class="btn" action="f"
disabled="option1.chosen">Execute</button-load>
```

Where `action` is a function to be used by the `button-load` directive and `disabled` is an expression to be evaluated by the directive (*on certain events*). Exceptions from the rule may happen, however.

22.1.11 File handling

File handling is associated with the following screens:

- **addfile**: serves for uploading/attaching new files
- **filelist**: displays all user's files; allows downloading, configuration and removal of the files
- **createnewtask**: during a task creation/configuration, a new file may be uploaded/attached and configured

To avoid repeating ourselves, we put a portion of a code serving for uploading/attaching files into a separate subcomponent, **common/fileinput**. It has a form of a directive, i.e. it consists both of an HTML template and a controller handling the logic behind.

File configuration is handled on several places (*all of the screens mentioned at this section*). We approached the problem by creating a subcomponent **common/filesettings**, consisting of a modal window with corresponding controls. On opening the modal, data associated with the existing file configuration is loaded from the server and sent upon close.

Last, but not least, **filelist** screen has a similar implementation to **taskconfigs** screen (*described in the following section*), consisting of a simple table, displaying basic information about each file, while providing actions to further manipulate the files (*configuring, downloading and removing the files*).

22.1.12 Task handling

Task handling is associated with the following screens:

- **createnewtask**: serves for creating new tasks while also allowing editing existing tasks
- **importtask**: allows creating new tasks by configuration import

- **taskconfigs**: displays all user's tasks, shows their basic information and allows for their basic manipulation
- **taskresult**: a comprehensive display of a task's result, provided by the server; allows sending feedback and reexecuting the task

createneutask consists, among others, of a **common/fileinput** component, to allow for a simple file upload / attach right during a task's creation / editing. Several form controls are available to allow a detailed task configuration.

importtask is a relatively simple screen consisting only of a text field and file input field. The selected task configuration file is processed via HTML5 file API and its data sent to the server (*upon clicking the corresponding submit buttons*).

taskconfig, similarly to **filelist** screen, is basically a table of tasks, displaying basic information, such as identifier, last modification date and description, while also providing several buttons for manipulating the tasks. The list of tasks, when obtained from the server, is handed to **pagination** directive. The directive processes the list, providing only a sublist to surrounding components, based on a currently selected page.

An important aspect of **taskconfig** is its ability to display each task's current state. Based on the state, different actions may be taken for each task state (*e.g. a running task may not be removed, while only a finished task has a 'go to result' button available*). The states are processed in the following manner:

- At the beginning, a server returns a list of all tasks, while also providing information about each task's current state.
- A list of tasks that are running, is created.
- Each of the task's state from the list is requested each 3 seconds via a time-out function. (*It is taken into a consideration that the server may take a longer time to respond, in which case the interval may be prolonged.*)
- A task, which is no longer running, is removed from the list.
- An action, such as re-running a task, may again change a task's state. If a task is this way put into the running state, it is added to the list of running tasks.
- Upon visiting a different screen, the time-out function is cleared.

22.1.13 Taskresult screen

The **taskresult** screen provides a comprehensive display of a task's result, which was previously computed by the server. The screen allows examining the result, provides means to store and send feedback to the server, and allows for downloading of exported result in various formats.

JSON data binding

In order to work properly, the application needs the following JSON objects from the server:

- `$scope.result` - represents an actual result of the algorithm. Additionally, the object is used as a binding variable for user changes.
- `$scope.feedback` - contains saved user changes from a previous iteration of the algorithm.
- `$scope.inputFile` - a user input file.
- `$scope.configuration` - contains information about chosen knowledge bases, primary knowledge base, etc.

Additional important objects include:

- `$scope.ignoredColumn` - represents ignored columns for feedback.
- `$scope.noDisambiguationColumn` - represents ignored cells in all columns.
- `$scope.noDisambiguationCell` - represents ignored cells.

An important utility object is "`$scope.locked`" with flags for locked/unlocked state of entities. Basically, every data change causes setting of a corresponding lock to "true" (*representing locked state*). It allows for sending only the actually modified data to the server.

In the beginning, all entities are unlocked. After a user makes some data modification and reruns the algorithm with his/her feedback, the modified entities will be locked. They stay that way until a user modifies his/her feedback again.

The structure of the object is as follows:

```
$scope.locked = {  
  tableCells: ...,  
  subjectColumns: ...,  
  graphEdges: ...,  
  statisticalData: ...  
}
```

- `tableCells` - two-dimensional array, first dimension ranging from -1 to number of rows (*-1 representing the header row*), the second dimension from 0 to number of columns.
- `subjectColumns` - one for each knowledge base (*at most one column can be chosen/locked*).
- `graphEdges` - for relation changes between two columns.
- `statisticalData` - for changes of data cube.

"\$scope.selectedPosition" determines selected position in the table of classifications/disambiguations, e.g.:

```
$scope.selectedPosition = {  
  column: 3,  
  row: -1  
}
```

"\$scope.selectedRelation" determines selected relation between two columns in the graphvis component, e.g.:

```
$scope.selectedRelation = {  
  column1: 2,  
  column2: 0  
}
```

The code of taskresult screen, being relatively complex, is divided into several sections:

- **classdisambiguation**: generates a table showing classifications and disambiguations suggested by the algorithm. The user can edit this suggested values.
 - **cdlock**: is a directive which shows lock/unlock icons. Also detects user changes.
 - **cdtable**: generates table using cdrow directive. The first row contains headers of the input file.
 - **cdrow**: represents a single row of the table. Shows a winner classification or disambiguation and provides means for editing the feedback.
 - **cdmodalproposal**: a modal window allowing a user to create and save his/her own classification/disambiguation for each cell in the table. A user can add his own entities (*but only to the primary knowledge base*).
 - **cdmodalselection**: a modal window showing detailed information about a picked cell from the table.
 - **cdselecting**: generates cdselectbox and cdsuggestion for all knowledge bases (see below).
 - **cdselectbox**: represents a select box using a component ui-selectbox, which is a smart version of select box that allows for displaying HTML elements inside its options (more on <https://angular-ui.github.io/ui-select/>).
 - **cdsuggestion**: allows to search for an appropriate classification or disambiguation via a label.

- **cdcheckboxes:** provides settings of ignored values
- **relations:** shows suggested relations using the graphvis component.
 - **rlock:** shows lock/unlock icons and detects user changes in the graphvis component.
 - **rmodalselection:** is a modal window providing details of a concrete relation.
 - **rmodalproposal:** allows to create and save a custom relation.
 - **rselectbox:** a select box similar to cdselectbox, but for relations .
 - **rsuggestion:** allows to search for an appropriate relation via a label.
 - **graphvis:** represents the graph of relations (see "Grapvis" below for further details).
- **subjectcolumns:** for setting a subject column for all knowledge bases.
- **statisticaldata:** setting dimensions and measures and its predicates using the table directive (see below) for data cube.
 - **table:** generates a table for given dimensions or measures. Allows to choose predicate in a similar manner as relations.
- **controlsbuttons:** contains control buttons, which allow to browse the result, save feedback and re-run the algorithm.
- **export:** buttons for exporting the data in a desired format.

LodLive

Odalic experimentally cooperates with project LodLive (<https://github.com/dvcama/LodLive>). The project provides means to browse resources and their related entities.

A copy of the project is lightly adapted to allow for communication with Odalic. An exit button was added as well as a button for returning a selected resource URL.

The communication between Odalic and LodLive is solved by HTML5 Message API (`Window.postMessage()`). For more information, please visit <https://developer.mozilla.org/en-US/docs/Web/API/Window/postMessage>.

Graphvis

A graphvis is a component that is a part of a relationship discovery step of the **taskresult** screen. Graphvis is basically just an SVG (*HTML5*) element with several controlling buttons.

At the beginning, a graph is created, based on the data in the task result obtained from the server. Graph is created by the principles of OOP, i.e. vertices, edges and labels are all objects

that may be further manipulated. Then, since the graphvis is implemented using D3 library (<https://d3js.org/>), *tick* function is called continuously by the D3 (*which represents a smallest step in the simulation, what graphvis in its principle is - a simulation*).

Several actions are mapped to events, such as clicking on an edge label, or dragging a node with a mouse. The main idea behind the graphvis are *states*: the only moving parts of the graph are nodes. These are moving only if attractive / repulsive forces are active and if the graph is not stabilized. We can turn the forces off, thus putting the graph into a static state. This serves us for allowing a user to create his/her own links between the nodes. (*The states are changed by clicking on the corresponding buttons.*) Additionally, the states are applied also when a node is fixed due to a user moving it around (*which may be further released by double-clicking on it*). This state is preserved even when changin between 'link creation' and 'node dragging' mode.

22.1.14 Knowledge base configuration handling

Task handling is associated with the following screens:

- **kbconfig**: serves either for creating new knowledge base configurations or editing the exsting ones
- **kbimport**: allows creating new knowledge base configurations by configuration import
- **kblist**: displays all user's knowledge base configurations, shows their basic information and allows for their basic manipulation
- **setproperties**: serves for defining new, or editing existing ones, predicates and classes groups

kblist is in many ways similar to other 'listing' screens, such as **filelist** or **taskconfig**. It consists of a table, pagination directive (*not to overwhelm the user with all of the configurations on 1 page in case of many configurations defined*), and several buttons to allow the manipulation of the configurations.

kbconfig is a rather more complicated screen consisting of several controls to allow detailed specification of a knowledge base configuration. **tabset** (*provided by the library uibootstrap*) groups relevant controls under common tabs and allows switching among them. A new directive, for specifying string arrays by a user, has been added, **cilistbox** (*e.g. for specifying skipped attributes in the 'search' tab*). To allow defining and editing predicates and classes groups, a new screen had to be added - **setproperties**. This led to a problem with data persistence: normally, when switching between screens, user-entered data is lost. To avoid that, we used **persist** service to keep the user-entered data while browsing **setproperties** screen and reload the data upon entering **kbconfig** again.

kbimport shares a very similar implementation with **importtask** screen.

23 UnifiedViews DPU implementation

The plugin was built according to instructions in [Creation of Plugins \(DPUs\)](#).

It uses [Jersey client](#) library to access the server [REST API](#). First it uploads the file provided as the input to the DPU under a unique ID and follows it by formatting it according to the DPU instance configuration.

The task configuration uploaded to the DPU is used as a template, where the actual file identifier is injected before the DPU sends the task configuration to the processing server. The file identifier injection exploits the fact that the configuration is a serialized RDF. It builds RDF model from it with UnifiedViews internal [Sesame library](#) and replaces the subject of the property that relates the file identifier to the task. Then it serializes the model again and sends it to the server under unique ID.

The execution is done in a manner similar to the web client: the DPU gives command to start it and then periodically polls the execution state until it is terminal. In case of success, the DPU requests export of the outputs and redirects the `InputStreams` from the response to `FileOutputStreams` that are defined for the outputs of the DPU. In case of failure or success with warnings, DPU extracts error message or messages from the response. Server responses are parsed by mapping them to objects of classes annotated with JAXB annotations, employing Jackson library integrated to the Jersey. Only in the case of warnings, it would be impractical to map the whole, potentially large and complicated result, so a partial mapping via [JsonCreator](#) is defined.

24 Possible extensions and improvements

24.1.1 Algorithm

24.1.1.1 Learning from the user feedback

In general, the algorithm should be able to learn from feedbacks provided by the users:

- When certain classification is marked as being wrong for file X, column C, then we can learn from that and in the future penalize such classification for documents with same/similar structure.
- The same for a chosen alternative, which could be prioritized in similar conditions.

- Use the already executed classifications/disambiguations/relations discoveries as a learning set of cases in supervised learning and try to deduce classifications/disambiguations/relations discoveries for the other cases based on that.

24.1.1.2 Different kinds of feedback

Apart from the feedback where user essentially overrules the algorithm, allow the user to provide a negative feedback, marking some resources as undesirable, but let the algorithm to try alternatives.

24.1.1.3 General performance

- Find a better balance between the context taken into account and the number of queries.
- Reduce the usage of web search API.
- Enable parallel processing of interpreters and cooperative interruption of tasks.

24.1.1.4 Relations discovery

- Try to suggest properties with the domain being equal to the concept classifying the subject column.
 - Also it should try to look into relations being already associated with the entities classified with the same class, e.g. if the algorithm knew that column is classified as Country, it should try to map existing properties with the domain Country to the columns with relation values - in other words it should try to find a column containing population, area (which define the Country as the domain).
 - The same for ranges.
- Integrate relations discovery in a better way - e.g. so that found relations (and their domain/ranges) influence the classification and may cause selection of a different class and then different disambiguations.
- For statistical data, the algorithm does not run relation discovery.
 - But modified version of relation discovery could be executed - it could search the range types, and use the fact that every such predicate is qb:dimensionProperty, qb:measureProperty.
- Taking into account distribution of the values when looking for the property.
 - E.g. if there is a column containing values such as "1.8", "2.2", "2.0", it probably is not weight of a person, but rather his or her height.

- Taking into account recommendations for relations based on the similarity (in terms of the structure) between processed files.
 - For example if file A contains relations X, Y, Z and it is similar (in terms of its structure) to file B, which contains relations X, Y, it is probable that file B also contains relation Z and such relation should be suggested.
- Taking into account recommendations for relations based on the fact that certain relations typically occur next to each other. E.g. When there are properties foaf:firstName and foaf:age, then there probably is also property foaf:surname.
- The algorithm may also take account that subject column in the CSV file may be also an object of some triple, not just the subject - so it makes sense to also look for inverse relations.
- Algorithm selects the best matching relation not just based on the comparison of the cell value and object of the triple in the knowledge bases, but also by comparing CSV column title and name/URI of the candidate predicate in the given knowledge base. Nevertheless, in case of two knowledge bases giving evidence for the given relation, the selected predicate should not be taken by just comparing the similarity of the property name and the column title (which may be misleading), but rather by consulting [Linked Open Data cloud](#) and selecting more widely used predicate for these situations.
- Detect common violations of the established vocabularies, such is not respecting domains and ranges of the properties, when changing the classifications of related columns.
- Relations when the primary key is split into two or more columns (in other words subject column does not represent the whole "primary key") currently is not supported.
- Enable of processing of sets of tables and detect relations across two or more tables (foreign keys, M:N tables).
- Explore algorithm behaviour in certain corner cases:
 - What if two columns are classified by the same class? How to introduce handling of self-relations, e.g. some person is another person's boss?
- Use the vocabularies to infer other relations.

24.1.1.5 Classification/Disambiguation

- There are still some issues with the user feedback to the classification/disambiguation results of the algorithm:
 - Cells with the same literal value share the same disambiguation, which is the first one resolved.

- Performance issues with respect to classification/disambiguation,
 - Too many queries to the knowledge bases during disambiguation caused by ineffective restriction of the searched entities in the knowledge bases. For example, the algorithm for disambiguation typically takes into account the context of the disambiguated cell, such as the row and column in the table the cell is part of. Nevertheless, there is no differentiation among the meaning of the other columns' cells forming the context. For example, if I would disambiguate name of the school, the information about 'locality' (state, country) is really important to reduce the number of entities probed in the knowledge bases and it would also increase precision.
- Too many false positives in case of lower evidence for the disambiguated cells/classified columns or CSV files providing low context for the classified columns/disambiguated cells.
- Take into account the distribution of values.
- When there is not enough evidence, do not produce the almost arbitrary classification or disambiguation, but rather produce no result.
- If you have a file which contains name and abbreviation, then the algorithm tries to associate the same class with the name and abbreviation at the same time. But in this case one of the columns should be chosen as the preferred one, and the other one to become a property of the first one.

24.1.1.6 Knowledge Bases

- Proper use of hierarchy of concepts.
- Improve the behaviour to always select the most specific concepts
- Support for GeoNames base and Wikidata.
- Do a rigorous evaluation:
 - Evaluate precision/recall/performance gain.
 - Evaluate how the precision/recall changed after proper use of hierarchies, other tweaks to KBProxy.
 - Evaluate how the performance improved after adjustments.
- Alternative labels are not searched when searching the bases.
- Provide more than one winning concept for disambiguation.

- Currently there is just one winning concept for disambiguation when e.g. DBpedia is used. It is not a desired behaviour, but it is so because of the way the queries for candidates are made.
 - We use exact string match first, and if no matching resource is found, we use regular expression. So in most cases, there will be one exact match.
 - Before with the now deprecated Freebase, the search API was more similar to a free text engine, hence we got many candidates. But DBpedia is a SPARQL database and therefore has the text matching comparatively limited.
 - Ideally, we should first build an inverted index of labels of all URIs in DBpedia, and use that instead of doing string matching on labels using SPARQL or we we could use an existing full-text index when available.
- Take into account the hierarchy of KB.
 - Follow the "subclassOf" relations and automatically assume that more generic classes are also candidates.
 - Unfortunately this relations have to materialized first.
- Currently the multiple KBs are handled as separate runs of the algorithm. It could be interesting to run the algorithm only once and handle multiple KBs in the KBProxy (return results from all of the configured KBs and merge results with the same URL).
 - At least allow parallel execution, but with respect to the shared resources (hard-drive and network access).

24.1.2 UI Improvements

- Introduce graph visualisation for data cube export.
- Prevent the roll-on effect in some less used browsers when the screens change.
- Add user administration module to the UI.
- Improve token generation and management:
 - Allow the user to overview the issued tokens.
- Use [OAuth](#) and external services to log the users in.
- Adapt [LodView](#) or other means to view the details of resources.

25 Project history

- **March 2016**
 - preliminary meetings
 - introduction of the team members to the problem and needed tools
 - discussion about the project scope
 - mailing list established
 - roles assigned
 - consultations with Ziqi Zhang
 - preliminary requirements gathering and architecture discussions
- **April 2016 - May 2016**
 - drafting of project proposal
 - collecting and analysing project requirements
 - possible scenarios
 - internal presentations:
 - existing knowledge bases
 - AngularJS
 - RDF tools and libraries
 - REST API
 - internal glossary established
 - practising of Git usage
 - time-schedule discussions
 - trying out the project wiki
 - further consultations with Ziqi Zhang
 - project management tools chosen

- **late May 2016**
 - overall architecture drafting
 - developing usage scenarios and their mock-ups
 - project proposal finalized
 - preparations for the 1st iteration
 - time estimations
 - tasks assignment
 - project guidelines
 - setting-up the environments
- **1st iteration (June 2016 - July 2016)**
 - re-balancing of responsibilities among the team members
 - basic code structure for the UI and server with REST API created
 - works on substitution of deprecated Freebase with DBPedia KB
 - drafting the approach to data cubes handling
 - prototype allowing to upload a file, launch the TableMiner+ algorithm through a web client and display the results, using the REST API created
 - 2nd iteration planning
 - establishing of user stories too simplify testing and verification of satisfied requirements
- **2nd iteration (July 2016 - August 2016)**
 - relations support added, including visualisation
 - adjustments to file upload dialog
 - discussion about the possibility to browse the repositories
 - LodLive, LodView analyzed
 - draft of export according to the "CSV on the web"
 - support for multiple KBs added
 - feedback to classification and disambiguations added to the UI, but not taken into account yet by the server
 - re-factoring of API to better accommodate feedback process
- **3rd iteration (September 2016 - November 2016)**

- extensive bug-fixing of the features introduced in the previous iteration
- feedback integration to the algorithm
- removal of certain annotations as part of the feedback discussed
- 4th iteration planning
 - custom concepts provision by the user
 - task configuration export and import
 - tasks management
 - UnifiedViews integration
 - remote files access
 - statistical data draft finalized
 - API error handling
- **4th iteration (November 2016 - December 2016)**
 - UI re-factoring
 - testing on data from the Austrian catalogues
 - feedback to relations added
 - improving the resource search and introducing custom resource proposal
- **5th iteration (January 2017)**
 - paging of listings
 - URI prefix mapping support added
 - configuration import and export solved
 - multiple users support and authentication/authorization
 - server state persistence
 - statistical data export implemented
- **6th iteration (February 2017)**
 - bugfixing
 - UnifiedViews plugin created
 - KB proxy performance tweaking
 - documentation finalization
 - minor UI tweaks

- **7th iteration (April 2017 - May 2017)**

- feedback from the project defence incorporated
- runtime knowledge base proxies configuration implemented
- datatype relations omission bug resolved
- multiple subject columns support added
- UI tweaks
- Docker installation option provided
- proxy definitions exports and imports implemented
- logging fixed
- minor bugfixes

26 INSTALLATION DISC CONTENTS

- **ODALIC**
 - **Odalic Semantic Table Interpretations**
 - *Deployable* (contains the WAR deployable to Apache Tomcat)
 - *Javadoc* (site directory and JAR with generated Javadoc)
 - *Sources* (zipped release source files)
 - *Working directory* (server working directory template)
 - **Odalic UnifiedViews Plugin**
 - *Deployable* (contains the plugin bundle and a lib directory with OSGi dependencies)
 - *Sources* (zipped release source files)
 - **Odalic User Interface**
 - *Deployable* (contains odalic-ui and modified LodLive directories to deploy to server)
 - *Sources* (zipped release source files)
 - **Project documentation**

27 LOGOS

Variant A



Variant B

Open Data Linker and Classifier

Variant C



Variant D

ODALIC

Variant E



Variant F

