Decision Support System Code Guide

Application to Predictive Maintenance Systems Design

Hugo MUNOZ HERNANDEZ

Juan Jose MONTERO JIMENEZ

Rob VINGERHOEDS

July, 2021

Contents

1 Installation and configuration

For the installation procedure described here it is assumed that the user has installed a version of *Eclipse* that supports at least *Java* 8. Under the previous condition, the following steps may be followed to ensure the correct installation:

1. Download the *.zip* compressed folder of the project and save it in a selected local address in your computer.
2. Make right click over the compressed folder and use *Extract Here*, a decompressed folder with the same denomination has been created. Inside this folder, another folder named *InternshipProject* may be found.
3. Open *Eclipse* and browse to select *InternshipProject* as the working folder.
4. Once *Eclipse* has been initialized, go to *File* → *Import* → *General* → *Existing Projects into Workspace* → *Browse* and select the folder *InternshipProject*. The project will be built in the current workspace of *Eclipse*. The recognition of the folder as an *Eclipse* project is possible because of the file *.project* in the same folder.
5. In the case that some referenced libraries (*.jar*) seem to be missed, right click on *InternshipProject* at the workspace menu on the left → *Build Path* → *Configure Build Path* → *Libraries* → click on *Classpath* in the list below and now the option buttons on the right are available. Delete the paths that are indicated as erroneous and click *Add JARs* on the left to reintroduce the libraries that are missed. Browse to the *external-libs* folder and select the *.jar* files that have to be restored, then just click OK and *Apply and Close*. However, this problem is not likely to occur as the *.classpath* file store the path to all the dependencies of the project. Another possible solution to the problem if this existed would be to open the *.classpath* file with a plain text editor and change the paths that are referencing a local folder in another user computer to a general path starting at the folder *external-libs* of the project.

Now that the code is installed, some configurations are needed to star execution. There is a class named *AppConfiguration* in the *User* package which only contains public static attributes and a couple of methods to build variables. These attributes are read and used by other classes of the project, so they are stored together in the accessory class *AppConfiguration* to be accessed (not modified) by other pieces of code in the application. The very first change that the user must make in the mentioned class before executing the code is setting the *data˙path* attribute to the actual local path of the *data* folder of the project. From that point, the attributes on *AppConfiguration* should be updated to the file names that are wanted to be used inside the *data* folder (*.csv, .owl,.prj*, etc). See the *javadoc* of the project and the comments in the source code of *AppConfiguration* to get known about the meaning of the attributes.

2 Basic usage

The usage of this application will be performed simply by the execution of java classes from Eclipse (or any other IDE). It may also require the manipulation of files (ontology files, *.csv*, *myCBR* project files, etc) in the designated file folders. Unless a modification of the source code is needed for some reason, there are 5 executable classes in the project that concern to the user at this moment: *CSVtoOntologyExec*, *myCBRSetting*, *SPARQL*, *GUI2*, *GUI3*, *OntologytoCSVExec* and *SWRLAPIexec*.

2.1 *SPARQL* queries

A simple executable class has been added to the project to execute *SPARQL* queries on the working ontology in the same way that they are available in *Protégé*. The *Jena* tool (see Section 3 for more information) is used for that purpose. The *SPARQL* queries allows the user to obtain an accurate required information from the ontology. A general reference for the *SPARQL* language can be found at [11]. The user just needs to write the query in a correct syntactical form and execute the class file to get the results of the query on the console. These queries are very quick in their execution as they do not need to run a reasoner and check the ontology consistency. Even if the current implementation in the project is accessory, the *SPARQL* queries could be potentially used to extract information from an ontology for the definition of ontological similarity values.

2.2 *SQWRL* queries

The implementation of the *SWRL API* together with the reasoning engine *drools* allows to add *SWRL* rules to the working ontology execute *SQWRL* queries. See Section 3 for more information. The *SQWRL* language, which is based on the *Semantic Web Rule Language* (*SWRL*), opens the possibility to accurate queries with a relatively simple syntax. There is a paper [21] where the creator of this language give an introduction to the main rules and basic syntax of the queries. An inconvenient has been found for the use the *SQWRL* queries for the application case in the current research project: the reasoner must be run once for each query, so, when working with ontologies having a big number of individuals declared, this process can require an important amount of *RAM* memory in the computer and it can last long if many queries are wanted to be executed. That is why, at this moment, the direct querying through the *OWL API* is used for extracting information from an ontology, as it only requires to run the reasoner (*HermiT*) once before executing as many queries as desired. Indeed, the example class in the project *SWRLAPIexec* allowing to execute *SQWRL* queries may not work correctly due to the compatibility problems of *SWRL API* with the latest version of *OWL API*. See Section 3. However, as it is potentially useful for the development of the project, the *SWRL API* implementation is still included in the project.

2.3 Load data in a table format (*.csv*) to an ontology file (*.owl*) using *CSVtoOntologyExec*

When executed, the class *CSVtoOntologyExec* will read the content in the specified data base in a *.csv* file and load the information into an ontology file (*.owl*). An important consideration is that the *.owl* files should be written in *RDF/XML* syntax (choose that option when using *Save as* in *Protégé*). In what concerns to *myCBR* retrieval, this class may only be executed when the case base information and similarity values contained in the *.prj* file should be changed or updated to perform CBR queries using new data. The path for working file folder should have been set in the *AppConfiguration* class, together with the corresponding files denominations. In this folder, it is recommended to locate a clean version of the ontology that is wanted to be used (without instances or individual or property assertions) and another copy (of course with a different name) where the data base will be stored. So, when updating the data base the procedure should be as follows:

- Delete the data base ontology file.
- Make a copy of the clean ontology file and change the name as desired . Of course, the same name should be specified in the *AppConfiguration* class (*ont_file_name*).
- Execute *CSVtoOntologyExec*, which will read the clean ontology file (specified in *AppConfiguration* as *base_ont_file_name*) and the data base of the *.csv* file (specified in *AppConfiguration* as *csv*) to merge the information in the ontology data base file.

The translation of the information stored in the table to ontological entities is stated with the appropriate using of the methods of the class *CSVtoOntology* in the package *OntologyTools*. See the *javadoc* of the project for details. By this way, the code is flexible to adapt to the ontological meaning of the content in the different columns and cells forming the table tabular data base.

In this particular case, the executable code in the *CSVtoOntologyExec* class is configured for the Predictive Maintenance data base and the *OPMAD* ontology. Nevertheless, the class could be modified if needed to suit to another different case or to adapt to a restructuring of the current data base and ontology.

2.4 Load data from an ontology file (*.owl*) to a table format *.csv* file using *OntologytoCSVExec*

The class *OntologytoCSVExec* is an executable class allowing to extract data from an ontology file and rewrite it in an organized table format into a *.csv* file. The class *OntologytoTabular* is used to get the data and organize it in a structured tabular *List* object. In the mentioned class, a *Jena API* implementation is used to be able to execute a series of *SPARQL* queries on the working ontology and get a tabular data as a result. The advantages of this type of queries for this application are their execution time (fast execution as they do not need reasoning) and the retrieval results obtained on the shape of organized tables. Now, one important issue should be underlined: the *SPARQL* queries have to be adapted to the particular ontology of the user, as well as the organization criteria for the data when stored as a *.csv* table. Even if it is possible to use one single query for the data extraction, it is recommended to divide the query in various of them when the size of the database is remarkable, as it would be much faster to execute. The list of queries is automatically built by a method in the *AppConfiguration* file. It is required to specify the static *String* parameter *queryHead* in the mentioned class to set the first part of the *SPARQL* queries, that will be common to all the queries in the list. This part of the query aims to define the variables that are going to be retrieved and to initialize the selection block. The closing of the query is also common to all the queries in the list, so it can be specified with the parameter *queryEnd*. The body commands of the queries, which will use the relations existing in the ontology to extract the data, are listed in the parameter *queryBodyList* in the *AppConfiguration* file. Then, the method *queryList* will build the strings with the full queries adding one head and one closing to each one of the body commands blocks. A reference to the *SPARQL* syntax can be found at [11]. The general procedure to determine the query structure is to choose the variables defining the cases, establish which entities in the ontology contain that information and look for properties or ontological relations that build links between the different pieces of data.

The output data of a *SPARQL* query is organized in columns , one for each requested variable. Cells in the table can store one single data value. If a variable value in one of the columns matches more than one value of other variable, then there will be one row in the query results table for each combination of values. That comes to say, using a simplified example, that having a table with only two columns in which each one of the n values of the first variable matches two values of the second variable, then the table will have $2n$ rows, where the values of the first variable variable will appear twice in the column. The same logic extends to several variables with multiple values, that would imply to add as many rows as necessary to include all the combinations of values. An example for the application case of Predictive Maintenance Decision-Support System is shown in Figure 1, where it is observed that there are as many rows concerning the reference index '1' as values of the field *Input type* existing for such case. Note that the indices are ordered using the ontological *Protégé* criteria instead of mathematical order.

Reference	Publication_Year	Task	Case_study	Case_study_type	Input_for_the_model	Input_type
"1"	2019	Health modelling	Simulated_jet-engines_data	Rotary_machines	Time_series	Baypass_ratio
"1"	2019	Health modelling	Simulated_jet-engines_data	Rotary_machines	Time_series	Temperature
"1"	2019	Health modelling	Simulated_jet-engines_data	Rotary_machines	Time_series	Fluid_Pressure
"1"	2019	Health modelling	Simulated_jet-engines_data	Rotary_machines	Time_series	Spinning_speed
"10"	2016	One_step_future_state_forecast	Proton_exchange_membrane_fuel_cell	Energy_cells_and_batte	Time_series	Voltage
"100"	2013	Multiple_steps_future_state_forecast	Lithium-ion_battery	Energy_cells_and_batte	Time_series	Impedance
"100"	2013	Multiple_steps_future_state_forecast	Lithium-ion_battery	Energy_cells_and_batte	Time_series	Temperature
"100"	2013	Multiple_steps_future_state_forecast	Lithium-ion_battery	Energy_cells_and_batte	Time_series	Voltage
"100"	2013	Multiple_steps_future_state_forecast	Lithium-ion_battery	Energy_cells_and_batte	Time_series	Current

Figure 1: Example of SPARQL query results table for the application Predictive Maintenance Decision-Support System.

Additionally, a complete *SPARQL* query for the previously mentioned application case is included in Figure 2 as example to be modified by the user. There is an important detail that must be clarified in what concerns to the variable designations in the *SPARQL* query: as the characters blank space, '-' and '/' are not allowed, they are substituted respectively by '_', double '_' and triple '_'. When rewriting data in a tabular shape using the class *OntologytoTabular*, the headers of the table containing the variables designations are automatically recovered following the inverse transformation rule.

To optimize the execution time of the queries, they can be divided. Individual queries can be executed to obtain partial data tables in which the reference indices column will be extracted together with one or more of the other columns. It is necessary that all the partial queries get the reference indices column as the first column of their results table, so as later all the partial tables can be merged in one. Hence, in most of the cases the columns can be extracted individually, but sometimes it could be required that some columns are extracted through the same query. When two or more columns that store multiple elements in their cells have an order dependency between them, they must be associated to the same query. The order dependency means that the elements listed in the cells of one of the columns are linked to another corresponding element among the ones listed in the cell of another columns. So, the elements in both columns must be listed in the same order to preserve the relation between the elements of data.

Once defined the queries, the results coming from their execution will be rewritten to tables where each row should match only one case, merging the rows coming from the *SPARQL* result to gather in the correspondent cells the set of values for those fields that are multiple-valued. A reference column is chosen (first column) with integer reference indices so as the content of the cells of all the rows with the same index is merged. The general criteria to do so is that no value is repeated in the multiple values cells. However, a set of variables names can be listed using the parameter *Repetition_allowed* for those columns in which repeated values in the same cell should be allowed. As blank spaces are forbidden in the *SPARQL* syntax, when the text values are recovered they are rewritten substituting '_' characters by blank spaces, unless the user specifies no to do so via the parameter *Not_Spaced*. At the end, all the tables will be put together according to the reference indices column, so as the data coming from the list of queries is gathered into one single data table.

Then, here are the features that must be set up in order to define the data transference from the ontology to a *.csv* file:

- *SPARQL* queries list that will determine what information is requested to the ontology and how it is organized. To be set up in the *AppConfiguration* file.
- Parameter list *Not_Spaced* to be specified in the class *AppConfiguration*. In this list it must be included the designation tags of those variables or fields in the table where blank spaces are not desired or expected. For those columns of the table, the text value will be written as it is in origin in the query results, without replacing characters '_' by blank spaces.
- Parameter list *Repetition_allowed* to be specified in the class *AppConfiguration*. In this list the user must include, among those variables (columns) that could contain multiple elements in the same cell, in which of them it is allowed that elements are repeated more than once in the same cell.

For the example below, concerning the Predictive Maintenance Decision-Support System application, it is illustrated the structure of the *SPARQL* query (Figure 2), the results for a case with multiple values (Figure 3) and the final format in the *.csv* table. So, when the results are retrieved from the *SPARQL* query, more than one row may be obtained for each case to get all the values combinations of all the fields that are multiple-valued. In the Figure 3 it is observed how the case with reference index '8' has multiple values for the fields *Input type* and *Models*, while the description tags of the field *Model Type* are matched to particular values of *Models*. In this example, a single query has been used in order to clarify the explanation, but in the real application case the query was divided in parts. Indeed, a query was used for each of the columns in the table except the pair *Model Type* and *Models* and the pair *Performance indicator* and *Performance*, which were respectively included in the same query. The reason for that is that they must keep an order relation for the elements that are listed in the cells. The *Model Type* values must be stored in the same order that the corresponding elements in the column *Models* that they are describing. Same condition is needed for the elements listed in *Performance indicator* and the their value in the column *Performance*. The column *Model Type* is declared in the list *Repetition˙allowed*, so as more than one of the *Models* concerning one case could be described with the same type if they belong to the same family of models.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
                        PREFIX owl: <http://www.w3.org/2002/07/owl#>
                        PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
                        PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
                        PREFIX def: <http://www.semanticweb.org/j.montero-jimenez/ontologies/2021/2/OPMAD#>
                        PREFIX obo: <http://purl.obolibrary.org/obo/>
                        PREFIX cco: <http://www.ontologyrepository.com/CommonCoreOntologies/>
                        SELECT ?Reference ?Publication_Year ?Task ?Case_study ?Case_study_type ?Input_for_the_model ?Input_type ?Model_Type
?Models ?Online__Off___line ?Performance_indicator ?Performance ?Complementary_notes ?Study_title ?Publication_identifier
                                WHERE {

           ?a rdf:type def:Predictive_maintenance_system_module .
           ?a  obo:RO_0010002 ?d .
           ?b  obo:RO_0010002 ?d .
           ?b rdf:type def:Predictive_Maintenance_Article .
           ?Models rdfs:subClassOf  def:Predictive_maintenance_model .
           ?d rdf:type ?Models .
           ?e rdf:type def:Predictive_maintenance_case .
           ?e cco:designates ?a .
           ?e def:has_text_value ?Reference .
           ?b def:has_publication_year ?Publication_Year .
           ?Task rdfs:subClassOf def:Predictive_maintenance_module_function .
           ?h rdf:type ?Task .
           ?a def:has_predictive_maintenance_function ?h .
           ?a obo:BFO_0000051 ?j .
           ?j rdf:type ?Case_study .
           ?Case_study rdfs:subClassOf def:Maintainable_item .
           ?Case_study_type rdf:type def:item_type .
           ?b obo:RO_0010002 ?Case_study_type .
           ?Input_for_the_model rdfs:subClassOf def:maintainable_item_record .
           ?n rdf:type ?Input_for_the_model .
           ?a obo:BFO_0000051 ?n .
           ?Input_type rdf:type def:Data_variable .
           ?n obo:RO_0010002 ?Input_type .
           ?Model_Type rdf:type def:Model_type .
           ?Model_Type cco:describes ?d .
           ?a def:has_synchronization ?Online__Off___line .
           ?Online__Off___line rdf:type def:Module_synchronization .
           ?b def:has_title ?r .
           ?r def:has_text_value ?Study_title .
           ?b def:has_identifier ?s .
           ?s def:has_text_value ?Publication_identifier .
           ?Performance_indicator rdfs:subClassOf def:Performance_value .
           ?t rdf:type ?Performance_indicator .
           ?t cco:describes ?a .
           ?t def:has_text_value ?Performance .
           ?u rdf:type def:Complementary_notes .
           ?u cco:describes ?b .
           ?u def:has_text_value ?Complementary_notes .

                        }ORDER BY (?Reference)
```

Figure 2: Example of complete SPARQL for the application Predictive Maintenance Decision-Support System.

Reference	Publication_Year	Task	Case_study	Case_stud...	Input_for_the_...	Input_type	Model_Type	Models
"8"	2019	Remaining useful life estimation	Railway_track_geometry	Structures	Time_series	Plastic_strains	Data-driven	Bayes_model
"8"	2019	Remaining useful life estimation	Railway_track_geometry	Structures	Time_series	Mechanical_Pressure	Data-driven	Bayes_model
"8"	2019	Remaining useful life estimation	Railway_track_geometry	Structures	Time_series	Life_cycles	Data-driven	Bayes_model
"8"	2019	Remaining useful life estimation	Railway_track_geometry	Structures	Time_series	Mechanical_stresses	Data-driven	Bayes_model
"8"	2019	Remaining useful life estimation	Railway_track_geometry	Structures	Time_series	Elastic_strains	Data-driven	Bayes_model
"8"	2019	Remaining useful life estimation	Railway_track_geometry	Structures	Time_series	Plastic_strains	Physics-based	Particle_Filter
"8"	2019	Remaining useful life estimation	Railway_track_geometry	Structures	Time_series	Mechanical_Pressure	Physics-based	Particle_Filter
"8"	2019	Remaining useful life estimation	Railway_track_geometry	Structures	Time_series	Life_cycles	Physics-based	Particle_Filter
"8"	2019	Remaining useful life estimation	Railway_track_geometry	Structures	Time_series	Mechanical_stresses	Physics-based	Particle_Filter
"8"	2019	Remaining useful life estimation	Railway_track_geometry	Structures	Time_series	Elastic_strains	Physics-based	Particle_Filter
"8"	2019	Remaining useful life estimation	Railway_track_geometry	Structures	Time_series	Plastic_strains	Physics-based	Physics-based_model_for_track_settlement
"8"	2019	Remaining useful life estimation	Railway_track_geometry	Structures	Time_series	Mechanical_Pressure	Physics-based	Physics-based_model_for_track_settlement
"8"	2019	Remaining useful life estimation	Railway_track_geometry	Structures	Time_series	Life_cycles	Physics-based	Physics-based_model_for_track_settlement
"8"	2019	Remaining useful life estimation	Railway_track_geometry	Structures	Time_series	Mechanical_stresses	Physics-based	Physics-based_model_for_track_settlement
"8"	2019	Remaining useful life estimation	Railway_track_geometry	Structures	Time_series	Elastic_strains	Physics-based	Physics-based_model_for_track_settlement

(a)

Online__Off___line	Performance_indicator	Performance	Complementary_notes	Study_title	Publication_identifier
Online	Mean_absolute_percentage_error	"(<5%)"	"No_info_about_operationa	"A_knowledge-based_prognostics_framework_	"doi.org/10.1016/j.ress.2018.07.004"
Online	Mean_absolute_percentage_error	"(<5%)"	"No_info_about_operationa	"A_knowledge-based_prognostics_framework_	"doi.org/10.1016/j.ress.2018.07.004"
Online	Mean_absolute_percentage_error	"(<5%)"	"No_info_about_operationa	"A_knowledge-based_prognostics_framework_	"doi.org/10.1016/j.ress.2018.07.004"
Online	Mean_absolute_percentage_error	"(<5%)"	"No_info_about_operationa	"A_knowledge-based_prognostics_framework_	"doi.org/10.1016/j.ress.2018.07.004"
Online	Mean_absolute_percentage_error	"(<5%)"	"No_info_about_operationa	"A_knowledge-based_prognostics_framework_	"doi.org/10.1016/j.ress.2018.07.004"
Online	Mean_absolute_percentage_error	"(<5%)"	"No_info_about_operationa	"A_knowledge-based_prognostics_framework_	"doi.org/10.1016/j.ress.2018.07.004"
Online	Mean_absolute_percentage_error	"(<5%)"	"No_info_about_operationa	"A_knowledge-based_prognostics_framework_	"doi.org/10.1016/j.ress.2018.07.004"
Online	Mean_absolute_percentage_error	"(<5%)"	"No_info_about_operationa	"A_knowledge-based_prognostics_framework_	"doi.org/10.1016/j.ress.2018.07.004"
Online	Mean_absolute_percentage_error	"(<5%)"	"No_info_about_operationa	"A_knowledge-based_prognostics_framework_	"doi.org/10.1016/j.ress.2018.07.004"
Online	Mean_absolute_percentage_error	"(<5%)"	"No_info_about_operationa	"A_knowledge-based_prognostics_framework_	"doi.org/10.1016/j.ress.2018.07.004"
Online	Mean_absolute_percentage_error	"(<5%)"	"No_info_about_operationa	"A_knowledge-based_prognostics_framework_	"doi.org/10.1016/j.ress.2018.07.004"
Online	Mean_absolute_percentage_error	"(<5%)"	"No_info_about_operationa	"A_knowledge-based_prognostics_framework_	"doi.org/10.1016/j.ress.2018.07.004"
Online	Mean_absolute_percentage_error	"(<5%)"	"No_info_about_operationa	"A_knowledge-based_prognostics_framework_	"doi.org/10.1016/j.ress.2018.07.004"
Online	Mean_absolute_percentage_error	"(<5%)"	"No_info_about_operationa	"A_knowledge-based_prognostics_framework_	"doi.org/10.1016/j.ress.2018.07.004"
Online	Mean_absolute_percentage_error	"(<5%)"	"No_info_about_operationa	"A_knowledge-based_prognostics_framework_	"doi.org/10.1016/j.ress.2018.07.004"

(b)

Figure 3: SPARQL query results table for one particular case for the application Predictive Maintenance Decision-Support System.

Reference	Publication Year	Task	Case study	Case study type	Input for the model	Input type	Data Pre-proc	Model Approa	Model Type	Models
1	2019	Health modell	Simulated jet-engines data	Rotary machine	Time series	Temperature,	yes	Single model	Data-driven	Logistic regression
2	2019	Health assessi	Simulated jet-engines data	Rotary machine	Time series	Temperature,	yes	Single model	Data-driven	Logistic regression
3	2019	Remaining use	Simulated jet-engines data	Rotary machine	Time series	Health index	yes	Single model	Data-driven	OS-ELM (Online-secuential e
4	2019	Remaining use	Simulated jet-engines data	Rotary machine	Time series	Health index	yes	Multi model	Data-driven	KFOS-ELM (Kalma filter-base
5	2019	Remaining use	Simulated jet-engines data	Rotary machine	Time series	Health index	yes	Multi model	Data-driven	EOS-ELM (Esemble of OS-ELI
6	2019	Remaining use	Simulated jet-engines data	Rotary machine	Time series	Health index	yes	Multi model	Data-driven	AEKFOS-ELM (adaptive-weig
7	2019	Health assessi	Railway track geometry	Structures	Time series	Mechanical st	No	Multi model	Physics-based, Data-driven	Physics-based model for trac
8	2019	Remaining use	Railway track geometry	Structures	Time series	Mechanical st	No	Multi model	Physics-based, Data-driven, Physics-based	Physics-based model for trac

(a)

Online/Off-lin	Performance indicator	Performance	Complementary notes	Study title	Publication identifier
Off-line	Mean absolute percentage error	(<0.05)	1 operational mode, 10(	Aircraft engine	doi.org/10.1016/j.ast.2018.09.044
Off-line	Mean absolute percentage error	(<0.05)	1 operational mode, 10(	Aircraft engine	doi.org/10.1016/j.ast.2018.09.044
Off-line	Score function (the smaller the better), Mean accuracy, Error range	(58.96), (93.26), ([21,33])	1 operational mode, 10(	Aircraft engine	doi.org/10.1016/j.ast.2018.09.044
Off-line	Score function (the smaller the better), Mean accuracy, Error range	(35.78), (95.78), ([15,22])	1 operational mode, 10(	Aircraft engine	doi.org/10.1016/j.ast.2018.09.044
Off-line	Score function (the smaller the better), Mean accuracy, Error range	(34.56), (96.54), ([13,19])	1 operational mode, 10(	Aircraft engine	doi.org/10.1016/j.ast.2018.09.044
Off-line	Score function (the smaller the better), Mean accuracy, Error range	(33.13), (96.76), ([11,18])	1 operational mode, 10(	Aircraft engine	doi.org/10.1016/j.ast.2018.09.044
Off-line	N/A	N/A	No info about operatior	A knowledge-b	doi.org/10.1016/j.ress.2018.07.004
Online	Mean absolute percentage error	(<5%)	No info about operatior	A knowledge-b	doi.org/10.1016/j.ress.2018.07.004

(b)

Figure 4: Final results table as it is loaded in the *.csv* file for the application Predictive Maintenance Decision-Support System.

2.5 Preparing project case base and similarity values for *myCBR* with *myCBRSetting*

The class *myCBRSetting* may be executed when the data base in the *.csv* (file name specified as *csv* in the class *AppConfiguration*) file or the ontology *.owl* file have been changed (normally their modifications are coordinated). In order to perform *myCBR* queries on a new case base, or a case base that has been updated, the information must be written in the *.prj* (file name specified as *projectName* in the class *AppConfiguration*) file, which is the one that contains the project data for *myCBR*. Moreover, the class will run an *OWL* reasoner (more precisely the *HermiT* reasoner, see Section 3) on the ontology that will check its consistency and infer relations for the knowledge contained in the ontology. In consequence, the execution of the class can take a bit of minutes, depending on the ontology size. The main reason to use the reasoner is to be able to perform queries on the ontology to extract information that can be used to calculate ontological similarity values. Moreover, the *HermiT* reasoner allows to make queries about relations that are not initially explicit in the ontology but inferred during the reasoning process. Obviously, if the ontology is not consistent the execution will stop, so to work with an ontological data base it must be consistent. Furthermore, even if the data base can be loaded to the project file either from a *.csv* data file or from an *.owl* ontology, the ontology file is always necessary to obtain the similarity values. The class *myCBRSetting* may be modified to adapt the code to a new application other than the Predictive Maintenance Decision-Support System. This is because the ontology relations and the method for semantic similarity calculation depend obviously on the application.

When executing, at first, the class uses an instance of *CBREngine* to load the current *.prj* file, delete all the existing instances in the case base and introduce the new ones, with their corresponding attribute values, by importing the chosen *.csv* file or the *.owl* ontology file. If the *.owl* database is chosen, the procedure to extract the data is exactly the same as the one followed when executing *OntologytoCSVExec*, see the Subsection 2.4 for more details. The class *OntologytoTabular* will get the data from the ontology following the structure of the *SPARQL* queries list. Then, once the data has been tabulated, it is loaded directly into the *myCBR* project file instead of creating a *.csv* file. So, using the direct importing from the *.owl* file has the same result that creating the *.csv* file form the ontology and loading it with the default *myCBR* importer.

After that, the similarity functions are established with the appropriate values. In particular for the Predictive Maintenance Decision-Support System application, for the field *Task* associated to each one of the cases, the similarity values are calculated using an ontological method. The querying of the ontology uses the classes *DLQueryEngineIRI* and *DLQueryParserIRI* of the package *OnotlogyTools*. See the *javadoc* of the project for more details.

Case variable	Variable type (*myCBR*)	Values
Task	Symbol	Fault feature extraction, Fault detection, Fault identification, Health modelling, Health assessment, Remaining useful life estimation, One step future state forecast, Multiple steps future state forecast.
Case study type	Symbol	Rotary machines, Reciprocating machines, Electrical components, Structures, Energy cells and batteries, Production lines, Others.
Case study	String	The *myCBR* Levenshtein function is used. Similarity is calculated with the quotient $\frac{number\ of\ characters\ reference\ String - Levenshtein\ distance}{number\ of\ characters\ reference\ String}$
Input type	Symbol	A list of variables must be provided separated by ', ' and where all the words should begin with capital letters as it is established in the case base. An additional Levenshtein method in *Java* allows to support misspelling up to 3 erroneous characters.
Online/Off-line	Symbol	Online, Off-line, Both, Unknown synchronization.
Input for the model	Symbol	Signals, Structured text-based, Text based maintenance/ operations logs, Time series.
Publication Year	Integer	This field is not provided by the user, the application will used the current date automatically. The most recent cases in the case base will be prioritized over the older ones.

Table 1: Case variables for querying and retrieval

2.6 Query and retrieval using the *GUI*'s

Once executed *CSVtoOntologyExec* to load the data base into the ontology and *myCBRSetting* to configure the *.prj* file for *myCBR*, only the executable *GUI*'s are required to query and retrieve until the case base is wanted to be modified. The tool *myCBR* uses the case base, the attributes and the similarity functions specified in the *.prj* file to search the most suitable case for the given query. To visualize or to modify manually the project file the *myCBR Workbench* application may be used.

When executing any one of the *GUI's*, the class *Recommender* is used. Most of the similarity functions are defined during the execution of the class *myCBRSetting*, but there is one particular field in the Predictive Maintenance Decision-Support System which similarity values are defined by comparing the current query to all the cases in the data base individually, and that is *Input type*. An analog method is used, which is analog to the one applied to establish the similarity values between the different Predictive Maintenance functions (field *Task*).

Two executable *Graphical User Interfaces* are provided for querying: *GUI2* and *GUI3*.

2.6.1 *GUI2*

The *GUI2* allows the user to perform one query at a time by specifying the following parameters:

- Values of the case fields for the retrieval. Some of them must be typed (*Case Study* and *Input type*) and for the rest of the fields the values are selected form the ones available in a drop down menu.
- Value of the weights assigned to each field.
- Type of amalgamation function that is used to get the global similarity value of each case.
- Number of cases to be retrieved. The resulting list of cases (ordered from higher to lower similarity value) will be shown in the screen after submitting the query.

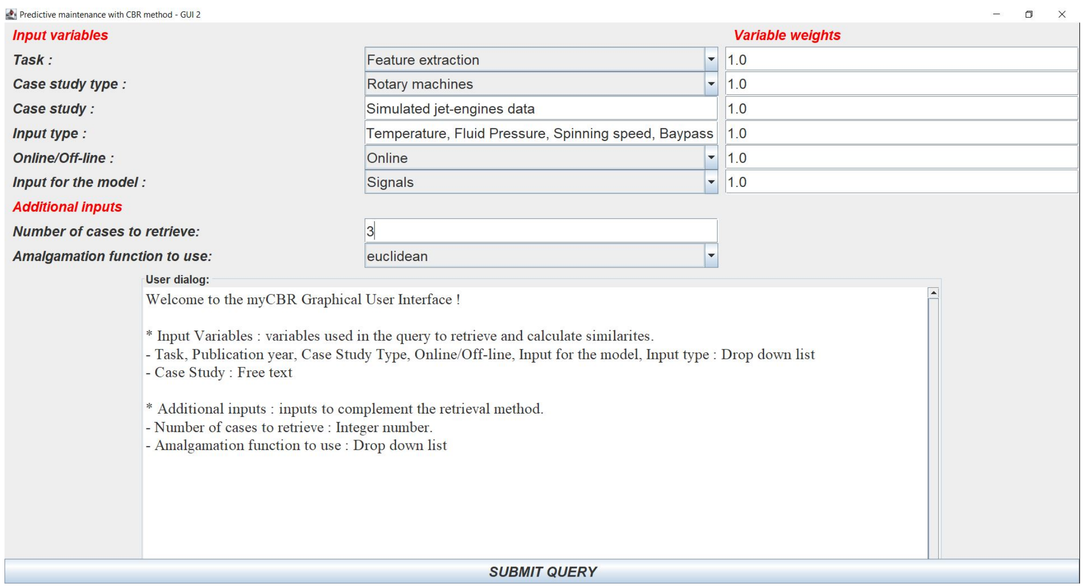

Figure 5: Window of the *GUI2*

If one of the fields is left blank, then its weight in the global similarity value is automatically set to 0. The value of the field *Case study* is expected to be equal to one already existing in the data base, otherwise the similarity value will be 0 for all the cases.

The field *Input type* contains a list of variables (most of them physical variables) that are considered in the Predictive Maintenance model of each case. This list must be typed with the terms separated by ', ' and all the words starting by capital letters, as they appear in the case base. Nevertheless, both fields can manage with possible misspelling errors using the Levenshtein distance method. In particular, the *Case study* field is defined as string type in *myCBR*, and it uses the default Levenshtein comparison function. But, for the *Input type* field, the Levenshtein method is not available in *myCBR* as it is declared as symbolic value. So, an additional method (class *LevenshteinDistanceDP*) to allow up to a distance of three erroneous characters in the spelling is implemented in the similarity function definition (*Recommender*). See the *javadoc* for more details.

2.6.2 *GUI3*

Using this *GUI*, the user is able to execute a list of consecutive queries provided in an input file in *.csv* format and to save their results in separate files (one for each query in the list). It is necessary to prepare an input file with the appropriate structure (see Figure 6). An example of input file is also provided in the actual project folder of the application. For the fields that are stated as symbolic in *myCBR* and for the string variable *Case study*, the user must type in the input file a value which is included among the possible values for each field (see Table 1), otherwise the similarity will be just 0. Symbolic fields, with the exception of *Input type*, do not support misspelling. The field *Input type* contains a list of variables separated by ', ' where the words should begin by capital letters (as they are in the data base). As soon as one of the variables of the list exist in one of the cases in the data base the similarity value will not be 0 for that case.

Task	w1	Case study typ	w2	Case study	w3	Online/Offline	w4	Input for the model	w5	Input type	w6	Number of ca	Amalgamation function
Fault detectio	1	Rotary machi	1	Simulated jet-	1	Online	1	Time series	1	Temperature	1	20	euclidean
Feature extra	1	Rotary machi	1	Simulated jet-	1	Offline	1	Time series	1	Temperature	1	1	euclidean
Fault detectio	1	Rotary machi	1	Simulated jet-	1	Offline	1	Time series	1	Temperature	1	5	euclidean

Figure 6: Example *.csv* input file for the *GUI3*

After having prepared the input file, the *GUI* window will just require to the user to provide the name of the input file and also that of the result files, as shown in Figure 7. For each one of the queries in the list, a result file will be generated with the denomination specified by the user and an index added to the name indicating its position in the list of queries. An example result file is shown in the Figure 8. The content of the result files is another list containing the information about the cases that have been retrieved (as many as demanded by each query).

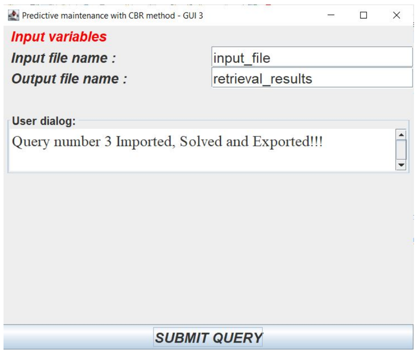

Figure 7: Window of the *GUI3*

Reference	Sim	Task	Case study typ	Case study	Online/Off-lin	Input for the r	Model Approa	Models	Input type	Number of inp	Performance	Performance	Complementa	Publication identifier
209	0,919	Fault detectio	Rotary machi	Simulated jet-	Online	Time series	Single model	Voting metho	Spinning spee	3	N/A,	N/A,	No info about	doi: 10.12700/APH.15.1.2018.2.10
191	0,901	Fault detectio	Rotary machi	Simulated jet-	Online	Time series	Multi model	Bayes model,	Measurement	2	Robustness , I	N/A,	10 operationa	doi :10.1109/TCST.2011.2177981
166	0,845	Fault detectio	Rotary machi	Wind turbines	Off-line	Time series	Single model	Gaussian proc	Temperature	22	Accuracy,	<0.945,	one case of s	https://doi.org/10.1016/j.renene.2018.10.088
167	0,845	Fault detectio	Rotary machi	Wind turbines	Online	Time series	Single model	Gaussian proc	Wind power	22	N/A,	N/A,	one case of s	https://doi.org/10.1016/j.renene.2018.10.088
210	0,837	Fault identific	Rotary machi	Simulated jet-	Online	Time series	Single model	Expert system	Results of a fa	1	N/A,	N/A,	No info about	doi: 10.12700/APH.15.1.2018.2.10
211	0,818	Fault detectio	Rotary machi	Simulated jet-	Off-line	Time series	Multi model	Hybrid Kalmar	Compressor T	8	Visual indicat	N/A,	6 operation m	doi: 10.1109/ACC.2013.6580567
192	0,818	Fault identific	Rotary machi	Simulated jet-	Online	Time series	Multi model	Bayes model,	Measurement	2	Robustness , I	N/A,	10 operationa	doi :10.1109/TCST.2011.2177981
195	0,786	Fault detectio	Rotary machi	Rotor shafts	Online	Time series	Multi model	Gauss-Markov	Imbalance sou	2	N/A,	N/A,	No info about	doi.org/10.1117/12.475502
72	0,772	Fault detectio	Structures	Tank reactor	Online	Time series	Multi model	Updated Rule	Temperature,	3	Probability,	From graphics	2 operational	doi:10.1016/j.ejor.2010.03.032
98	0,760	Fault detectio	Rotary machi	Steam Turbine	Online	Signals	Single model	Extreme Grad	Condenser va	7	N/A,	N/A,	No info about	doi.org/10.1016/j.microrel.2013.03.010
59	0,758	Remaining use	Rotary machi	Aircraft bearir	Online	Time series	Multi model	Relevance vec	Health index,	1	Score functio	19,66,	No info about	doi.org/10.1016/j.ress.2017.12.016
58	0,758	Remaining use	Rotary machi	Aircraft bearir	Online	Time series	Multi model	Ensemble lear	Health index,	1	Score functio	7,8,	No info about	doi.org/10.1016/j.ress.2017.12.016
60	0,758	Remaining use	Rotary machi	Aircraft bearir	Online	Time series	Single model	Particle filter,	Health index,	1	Score functio	301,8,	No info about	doi.org/10.1016/j.ress.2017.12.016
204	0,756	Remaining use	Rotary machi	Wind turbines	Online	Time series	Single model	Geolocation p	Distance, Deg	2	Prognostic ho	(65),(0.7),	No info about	doi.org/10.1016/j.renene.2017.05.020
2	0,755	Health assess	Rotary machi	Simulated jet-	Off-line	Time series	Single model	Logistic regres	Temperature,	5	Mean absolut	(<0.05),	1 operational	doi.org/10.1016/j.ast.2018.09.044
1	0,755	Health modell	Rotary machi	Simulated jet-	Off-line	Time series	Single model	Logistic regres	Temperature,	5	Mean absolut	(<0.05),	1 operational	doi.org/10.1016/j.ast.2018.09.044
51	0,754	Remaining use	Rotary machi	Simulated jet-	Off-line	Time series	Multi model	LSTM (Long-Sl	Temperature,	14	Root mean sq	6,9±4,7,	6 CMPASS dat	doi.org/10.1016/j.ast.2019.105423
53	0,754	Remaining use	Rotary machi	Simulated jet-	Off-line	Time series	Single model	Recurrent Ne	Temperature,	14	Root mean sq	10,2±5,8,	6 CMPASS dat	doi.org/10.1016/j.ast.2019.105423
52	0,754	Remaining use	Rotary machi	Simulated jet-	Off-line	Time series	Single model	LSTM (Long-Sl	Temperature,	14	Root mean sq	8,2±5,7,	6 CMPASS dat	doi.org/10.1016/j.ast.2019.105423
54	0,754	Remaining use	Rotary machi	Simulated jet-	Off-line	Time series	Single model	Gated recurre	Temperature,	14	Root mean sq	10,0±6,0,	6 CMPASS dat	doi.org/10.1016/j.ast.2019.105423

Figure 8: Example *.csv* result file for the *GUI3*

2.7 The *javadoc*

A complete *javadoc* has been generated for this project. It is available in the folder *javadoc* of the project. The easiest way to access to the documentation is opening the *index* file in the mentioned folder. It is an *HTML* file in the standard format for *javadoc* resources available in the web, and it may be opened with a web browser. The documentation contains a detailed descriptions of all the classes and methods organized in linked pages, allowing to navigate through the structure of the code. To update the documentation using *Eclipse*, go to *Project* → *Generate Javadoc*. When the window is open, the user must select the folder where the *javadoc* will be saved. It is recommended to use the folder *javadoc*, after having deleted the previous content to avoid problems. Of course, the generation of *javadoc* depends on the *javadoc* format comments that have been added to the code. See [12] for more details on the *javadoc* comments format.

2.8 Management of the *data* folder

The purpose of the *data* folder in the project is to store all the files that are needed for the execution tasks of the application. It is very important to remember that the denomination of the files that are wanted to be used inside this folder must be in coordination with the names introduced in *AppConfiguration*, so as the code is able to find such files. Some of the main types of files and their function are listed:

- Files *.owl* : the ontology files. One important consideration is that these files must be written in the *RDF/XML* syntax (choose that option when using *Save as* in *Protégé*). They can be opened with *Protégé* to visualize and edit the ontology or with a simple plain text editor, which allows to modify manually the statements. Two files may be used, one file should contain a clean ontology (without the declaration of the individuals in the data base) and the other one with all the data base loaded (and a different name, of course). An ontological data base could be directly used for the *myCBR* project build-up by selecting the appropriate option (user text input) when executing the class *myCBRSetting*. In any case, an ontology structure is required, either formed from a *.csv* table or directly provided by the user.

- Files *.csv* : files with a table format. A data base in *.csv* format may be needed to be loaded in the working ontology. The input and result files used by the *GUI3* are also *.csv* format. These files can be opened with *Excel*. Using a plane text editor to open *.csv* files may be recommended to ensure that no weird characters have been introduced by *Excel* at the beginning of the data (an unusual bug in *Excel* that may alter the name of the first column of data).

- Files *.prj* : the project files of *myCBR*, where all the information concerning the query and retrieval must be stored. This type of file can be opened with the *myCBR Workbench* application.

- Files *.ttl* : these files contain the ontology dependencies (BFO, CCO ontologies, etc). They are necessary to read the ontologies when working with local imports, so as the ontology is independent of the online servers. They are available at the GitHub repository [22], and may be updated with newer versions from time to time. For the execution of the code, some mappers are set using *OWL API* to link the *.ttl* files with the *URI* that are used by the ontologies to invoke the corresponding dependencies. For the *OPMAD* ontology, the following files are required: *ArtifactOntology.ttl, EventOntology.ttl, ExtendedRelationalOntology.ttl, GeospatialOntology.ttl, InformationEntityOntology.ttl, ro-import.ttl, TimeOntology.ttl.*

- File catalog-v001.xml : this file stores the paths to allow *Protégé* to import the local ontology dependencies in *.ttl* format when opening an ontology file that needs those dependencies.

3 Dependencies

Here are listed the main *APIs* and libraries which are needed for the application, which are included in the *external-libs* folder of the project:

- *OWL API* : it is an *API* that allows to read, modify, manipulate and create *.owl* ontologies. The version currently used in the application is 5.1.9, but future updates could be possible. The *API* is implemented by including the necessary *.jar* files in the *external-libs* folder of the application. The last version of *OWL API* is available at the *Maven* repository [2] (artifact owlapi-distribution), but it has been directly obtained as *.jar* files from [1] with all dependencies. Previous versions are also available. A complete documentation can be accessed at [3] and there is an introduction tutorial written by the creators at [6]. The licenses concerning the *API* are the Apache License [4] and the GNU license [5].

- *HermiT* reasoner: it is a reasoner that works in *Protégé*, the most used software for ontology manipulation. There is and implementation for *Java* based on the *OWL API*. The version used is the latest one (1.4.5.519), and it can be found at the *Maven* repository [7] or downloaded from [8] with all dependencies. The reasoner is needed to perform queries on ontologies through *Java*. The documentation for a previous equivalent version (1.3.8.4) is available at [9]. The GNU license es applicable [5].

- *myCBR*: it is an open source tool for case-based reasoning applications. The *Software Development Kit* of *myCBR* project has been implemented in the application with the appropriate *.jar* file. The *myCBR Workbench* application may be very useful for visualizing *.prj* files. All the information concerning *myCBR* project (source code, installation guide, tutorials, javdoc, etc) is available at the website [10].

- *Jena*: it is a tool for ontology manipulation. In paritcular, it is used in this application for adding the capability to perform *SPARQL* queries on ontologies through *Java* with the *SPARQL* executable class in the project, that uses the *Jena* methods. The latest version is used (4.0.0), and it can be found at the *Maven* repository [13] (artifact jena-arq) or downloaded from [14] with all dependencies. A complete documentation of the *Jena Core* is available at [15]. The Apache license [4] is applicable.

- *SWRL API* with *drools* rule engine: the *SWRL API* may be used to implement *SWRL* rules in an ontology through a *Java* application. Moreover, if the *drools* reasoner is added, the application is able to execute *SQWRL* queries on the working ontology. In this case, the latest version of *SWRL API* (2.0.9) is used, which is available at the *Maven* repository [16]and at [17] for direct *.jar* download with all dependencies. A *javadoc* is available at [18]. Nevertheless, the compatibility of the *SWRL API* with *OWL API* is only guaranteed up to version 4.5.9, even if some later versions could be supported. Moreover, to query an ontology with the *SWRL API* and *SQWRL* language, an additional implementation of a reasoner is necessary: the implementation of *drools* engine is available for this purpose. Once again, it may be found at the *Maven* repository [19] or for direct *.jar* download with all dependencies at [20].

References

[1] JAR download (April 2021): https://jar-download.com/artifacts/net.sourceforge.owlapi/owlapi-distribution

[2] MVN repository (April 2021): https://mvnrepository.com/artifact/net.sourceforge.owlapi/owlapi-distribution/5.1.17

[3] OWL API javadoc (April 2021): https://javadoc.io/doc/net.sourceforge.owlapi/owlapi-distribution/latest/index.html

[4] Apache License version 2.0: https://www.apache.org/licenses/LICENSE-2.0

[5] GNU LESSER GENERAL PUBLIC LICENSE: http://www.gnu.org/licenses/lgpl-3.0.txt

[6] MATENTZOGLU Nicolas., PALMISANO Ignazio. *An introduction to OWL API.* University of Manchester, 2016.

http://syllabus.cs.manchester.ac.uk/pgt/2020/COMP62342/introduction-owl-api-msc.pdf

[7] MVN repository (April 2021): https://mvnrepository.com/artifact/net.sourceforge.owlapi/org.semantic

[8] JAR download (April 2021): https://jar-download.com/?search˙box=org.semanticweb.hermit

[9] HermiT 1.3.8.4 javadoc (April 2021): http://javadox.com/com.hermit-reasoner/org.semanticweb.hermit/1.3.8.4/overview-summary.html

[10] myCBR website (April 2021): http://mycbr-project.org/index.

[11] SPARQL syntax reference (April 2021): https://www.w3.org/TR/sparql11-query/

[12] *Javadoc* reference (April 2021): https://www.oracle.com/technical-resources/articles/java/javadoc-tool.html

[13] MVN repository (April 2021): https://mvnrepository.com/artifact/org.apache.jena/jena-arq/4.0.0

[14] JAR download (April 2021): https://jar-download.com/artifacts/org.apache.jena/jena-arq

[15] Jena Core 4.0.0 (April 2021): https://jena.apache.org/documentation/javadoc/jena/

[16] MVN repository (April 2021): https://mvnrepository.com/artifact/edu.stanford.swrl/swrlapi/2.0.9

[17] JAR download (April 2021): https://jar-download.com/artifacts/edu.stanford.swrl/swrlapi

[18] SWRL API Javadoc (April 2021): http://soft.vub.ac.be/svn-pub/PlatformKit/platformkit-kb-owlapi3-doc/doc/owlapi3/javadoc/overview-summary.html

[19] MVN repository (April 2021): https://mvnrepository.com/artifact/org.apache.jena/jena-arq/4.0.0

[20] JAR download (April 2021): https://jar-download.com/artifacts/org.apache.jena/jena-arq

[21] O'CONNOR Martin Joseph, DAS Amar. *SQWRL: A Query Language for OWL.* Stanford Center for Biomedical Informatics Research, 2009.

[22] GitHub repository (April 2021): https://github.com/CommonCoreOntology/CommonCoreOntologies