

ADOxx KPI WEB DASHBOARD: Documentation on Architecture and First Prototype



Project Title	:	KPI Web Dashboard for monitoring specific KPIs and Goals
Deliverable	:	D1.0
Type of Deliverable	:	DataSource Wrapper and KPI Model: Architecture and Prototype Implementation
Nature of Deliverable	:	Documentation on Implementation of Project
Dissemination Level	:	Public
Delivery Date	:	June 30, 2017
Name of student	:	Frederick Dodzi Agbleke
Author(s)	:	Frederick Dodzi Agbleke, Damiano Falcioni
Reviewer(s)	:	Robert Woitsch, Barbara Re
Organization	:	BOC Asset Management (Research & Development)
Associated University	:	University of Camerino
Type of University Program	:	Erasmus+ Traineeship 2016/2017

Abstract

A Key Performance Indicator (KPI) is a measurable value that demonstrates how effectively an organization is achieving key business objectives. Organizations use KPIs to evaluate their success at reaching targets.

D1.0 is the first deliverable of documentation KPI Web Dashboard project which is based on the ADOxx meta-modeling environment. This document describes the architecture of some components of the dashboard and as well the implementation of an initial prototype.

In the prototype part of this deliverable, the delivered code is briefly described. We start by detailing a step-by-step process on how to set up (i.e. the DataSource wrapper and KPI model parts of the web dashboard excluding the implementation of the Dashboard Web interface) by get the source codes, build and run the dashboard. It also describes how the dashboard can be used with an example data source (i.e. excel sheet).

Keyword list

KPI Web Dashboard, KPI Model, Datasource Wrapper, Data source, KPI, Goal, Algorithm, Web service, Web dashboard interface

Glossary, acronyms & abbreviations

KPI	Key Performance Indicator
JSON	JavaScript Object Notation
REST	Representational State Transfers
URI	Uniform Resource Identifier
HTTP	HyperText Transfer Protocol
graphRep	Graphical Representation
attrRep	Attribute Representation
NoSQL	Not Only SQL
MSSQL Server	Microsoft SQL Server

1. Introduction	6
1.1. Deliverable objectives	9
1.2. Deliverable Outline	9
2. KPI Model and the DataSource Wrapper Components	10
2.1. KPI Dashboard General Architecture	10
2.1.1. Component Diagram	10
2.1.2. Sequence Diagram	11
3. Development of the KPI Dashboard Components	13
3.1. The DataSource Wrapper Overview	13
3.1.1. Dashboard Maven project for implementing the DataSource Wrapper	13
3.1.1.1. Java Packages and constituent classes in the Dashboard project	13
3.1.1.2. MySQL Module Implementation	15
3.1.1.3. MSSQLServer Module Implementation	16
3.1.1.4. Excel Module Implementation	17
3.1.1.5. REST Service Module Implementation	18
3.1.1.6. TripleStore Module Implementation	19
3.1.1.7. JsonFile Module Implementation	20
3.1.2. Detailed Description of DataSource Wrapper Class	21
3.1.3. Implementing the DataSource Wrapper as REST Web Service	21
3.2. The KPI Model	22
3.2.1. ADOxx Experimentation Library as a default library	22
3.2.2. Defining Classes and Attributes for the KPI modeling language	22
3.2.3. Defining Graphical Representation for Classes and Notebook Attributes	24
3.2.5. Creating a User to access the Modeling Toolkit	24
3.2.7. AdoScript to retrieve Notebook Attribute values of the instances on a KPI model	25
3.2.8. Configuring “KpiDashboardUtils” in the ADOxx development toolkit	26
3.2.9. The Dashboard Helper	26
3.2.9.1. Methods in the Dashboard Helper	26
3.2.9.2. Configuring the Dashboard Helper with the KPI Model	28
4. ADOxx Training Sessions Dashboard Example	33

4.1. Prerequisite Tools and source codes	33
4.2. Setting Up The Training Session Web Dashboard	33
4.2.1. ADOxx Environment Implementations	33
4.2.1.1. ADOxx Training Session KPI Model	37
4.3. Description of objects and relevant attribute values of the Training Session KPI model..	38
4.2.2. Importing the Dashboard war into Eclipse IDE	49
4.1.3. Configuration and Visualization of the KPI model Result.....	50
5. Conclusions and Plans for Future Work	52
References	53

List of Figures

Figure 2. 1 KPI Dashboard Conceptualized Structure	8
Figure 2. 2 KPI Dashboard Component Diagram	11
Figure 2. 3 KPI Dashboard Sequence Diagram	12
Figure 2. 4 Dashboard Project to be imported in Eclipse	14
Figure 3. 1 Library Management settings	22
Figure 3. 2 Class Diagram of KPI Meta-Model	23
Figure 3. 3 Creating a Modeltype and adding Library attributes	24
Figure 3. 4 Example of a KPI Model showing objects and their relations	25
Figure 3. 5 Selecting a DataSource Type using the Dashboard Helper	27
Figure 3. 6 Setting Configuration parameters for the selected DataSource type	27
Figure 3. 7 Adding User Input	28
Figure 3. 8 Adding KPI Fields to be retrieved	28
Figure 3. 9 Create New attribute for DataSource class	29
Figure 3. 10 Setting a program call to procedure to execute the Dashboard Helper jar	29
Figure 3. 11 Adding the new attribute to the DataSource “attrRep”	30
Figure 3. 12 Button to execute Dashboard helper jar	30
Figure 3. 13 Creating KPI attribute and setting program call to execute Helper jar	31
Figure 3. 14 Button to configure KPI Fields with the Dashboard Helper	32
Figure 4. 1 ADOxx Library Management	33
Figure 4. 2 KPI Dashboard Library Import	34
Figure 4. 3 Creating a New User and associate with the corresponding Library	34
Figure 4. 4 File management for adding files to ADOxx database	35
Figure 4. 5 ADOxx External coupling to execute AdoScript file	36
Figure 4. 6 KPI Model Import	36
Figure 4. 7 KPI Model for the second Excel sheet available in the DataSource	37
Figure 4. 8 KPI Model for the first Excel sheet available in the DataSource	38
Figure 4. 9 DataSource object configured to select sheet 2 of the Excel data	39
Figure 4. 10 Trigger the Dashboard Helper to configure DataSource object	39
Figure 4. 11 Setting Dashboard REST Endpoint	39
Figure 4. 12 Selecting a DataSource type	39
Figure 4. 13 Configurations Parameters required for connecting and retrieving data	40
Figure 4. 14 Setting User Inputs (OPTIONAL)	40
Figure 4. 15 DataSource object with Excel sheet configuration parameters	41
Figure 4. 16 Single Session Attendance KPI	42
Figure 4. 17 Configuring Single Session Attendance KPI	42
Figure 4. 18 Yearly Attendance KPI	43
Figure 4. 19 Configuring Yearly Attendance KPI	43

Figure 4. 20 Average Yearly Attendance KPI	44
Figure 4. 21 Configuring Average Yearly Attendance KPI.....	44
Figure 4. 22 Group by Year Algorithm.....	45
Figure 4. 23 Configuration of Group by Year Algorithm	45
Figure 4. 24 Calculate Average Algorithm.....	46
Figure 4. 25 Configuring Calculate Average Algorithm	46
Figure 4. 26 Goals to be evaluated.....	47
Figure 4. 27 KPI Model for Excel sheet 1	47
Figure 4. 28 Triggering the AdoScript to Generate the model JSON	48
Figure 4. 29 Exporting the generated model JSON	48
Figure 4. 30 Eclipse IDE import of the Dashboard war file	49
Figure 4. 31 Deploying Dashboard project on Tomcat Application	49
Figure 4. 32 Web Dashboard User interface	50
Figure 4. 33 Uploading model Export file on to the Dashboard to visualize KPIs.....	50
Figure 4. 34 Dashboard widgets	50
Figure 4. 35 Dashboard Overview of the ADOxx Training Sessions Data	51

1. Introduction

The main objective of this project is to provide a KPI (Key Performance Indicator) Web Dashboard in order to create a visualization of data from organizations and monitor performance through the achievement or otherwise of defined objectives and targets according to the overall business goals. A KPI Dashboard enables businesses to create, manage and analyze data from KPIs. This KPI dashboard implementation supports the utilization of data from various data sources which will be managed by DataSource Wrapper with the necessary connection parameters set right to aid to retrieval of data. This document looks at how data sources such as Excel, MSSQLServer, MySQL, TripleStore, REST service and from a JSON file were created and added to the DataSource Wrapper. The Wrapper has an extensible characteristic which provides possibility for new data sources such as Microsoft Access, BigData, NoSQL databases, etc to be added.

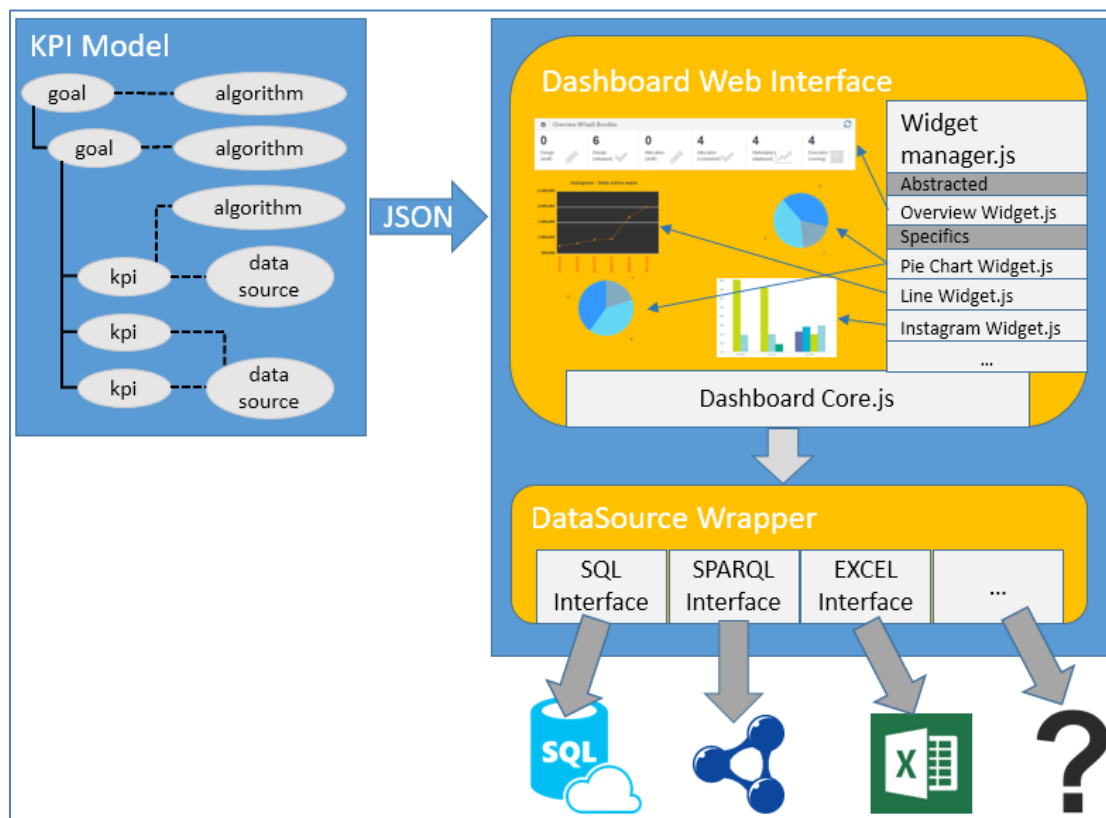


Figure 2. 1 KPI Dashboard Conceptualized Structure

The first part deals with a detailed explanation of the Data source Wrapper which serves as a data access layer is created from scratch.

The second part of the document explains how the project shows how the ADOxx modeling environment helps the user to model all aspects of a KPI where data provided by the model will

be evaluated based on some KPI values in order to assess the performance and achievement or otherwise of particular goals. The document attempts to explain how classes, attributes and their relations are created in the ADOxx development environment and with the help of AdoScript code (scripting language for ADOxx meta-model platform) created to give some functionality to the KPI model.

This part also explains how DataSource Wrapper (which is a Java component) provides data depending on which fields are required to provide a KPI value. The data source is selected by using connection parameters set in the KPI Model provided by the user for the particular data source to be used. This will aid students or developers who desire to build this dashboard project and subsequently improve on it to a fair idea how to start. It will actually provide a basis for future development in this area of KPI Dashboard development allowing them have more exposure with the ADOxx meta-modeling environment in addition.

1.1. Deliverable objectives

This document provides documentation and a prototype which will serve as basis for students and developers to understand how components of the KPI Dashboard are created. It intends to give start to further developments by interested developers who wish to work on projects with regards to organization KPI monitoring. It covers:

- A technical documentation for both the Data source Wrapper and the KPI Model
- A first implementation of the components of the KPI Dashboard and how they all work together to provide a first implementation of a working system.

1.2. Deliverable Outline

This document provides a detailed documentation and implementation of the KPI Dashboard and its component. It is divided into five (5) chapters with described as follows:

- Chapter 2 describes the methodology and the supporting tools that were used to create the various components of the KPI Web Dashboard. In particular, depicting the architectural framework, the functionalities and interactions of the individual components and sub-components.
- Chapter 3 attempts to give detailed description of how each component is created from scratch with particular focus on the KPI Model and the DataSource Wrapper components. It goes ahead to show how an example working Dashboard is created, with downloads of source codes, libraries and other files needed to build and run the system to give a better understanding of how the components work together to generate a final result on the dashboard.
- Chapter 4 introduces an implementation manual to assist users and developers to get the first example implementation of the KPI Web Dashboard.

- Finally, Chapter 5 concludes the deliverable, with a summary of its usefulness and contributions as well as outlines for future works and improvements.

2. KPI Model and the DataSource Wrapper Components

The main Web Dashboard interface which visualizes the data works in tandem with the KPI model. The model also connects to a data source through a web service call. Therefore it is important to have a technical understanding of these components and interactions between these components.

This chapter introduces an architectural overview of the above mentioned components while going into details where needed and also briefly describing the functionality of each component.

2.1. KPI Dashboard General Architecture

The architecture of the dashboard is composed of three (3) main components as already described above. It is made up the KPI model, the Data Wrapper and Web dashboard service including the interactions through the interfaces of these components. We use:

- i. A component diagram which describes the organization of physical software components in the system and the interfaces with which these individual components interact.
- ii. A sequence diagram and Use Case diagram to describe the flow of logic in a visual manner showing the behavior of the system.

2.1.1. Component Diagram

This systems architecture-level artifact depicts the high-level view of the software components and more importantly the interfaces to these components. This view helps in understanding how best the system can be organized especially during the development stage. The diagram shows the three main component interfacing with each other:

- The ADOxx environment component is also composed of two (2) sub-components (i.e. the development and modeling platform interacting to provide a basis for modeling KPIs).
- The KPI modeling components interfaces with the DataSource Wrapper by allowing connection to a selected data source to get data and use the data to provide relevant KPIs through evaluation algorithms. The diagram focuses on three (3) data sources configured in the DataSource wrapper. Other additional data source configuration will be provided later in this document.
- The web dashboard gets the output from the KPI model and generates a simple cockpit displaying different assessment of goals and KPIs. This document does not focus on the development of the Web Dashboard in detail since it is not part of the deliverables.

The figure below depicts the software components of the KPI Web Dashboard as described above:

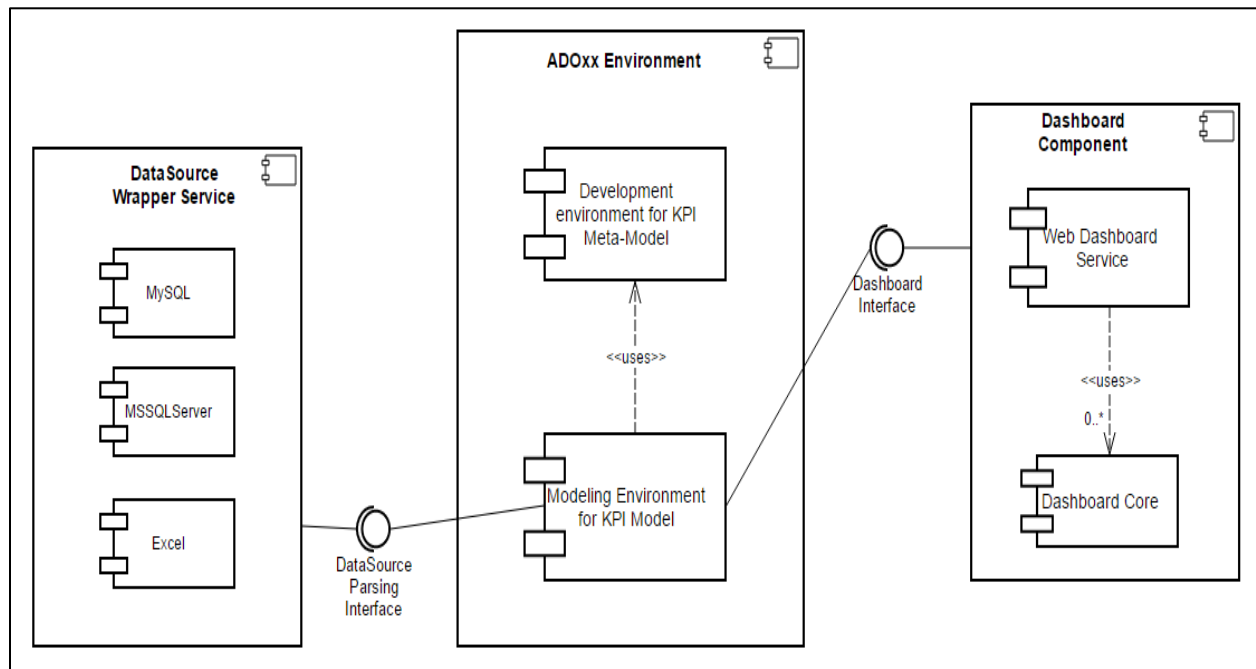


Figure 2. 2 KPI Dashboard Component Diagram

2.1.2. Sequence Diagram

The sequence diagram below describes the interaction among components of the KPI Web Dashboard in terms of an exchange of messages or data showing communication between the components over time. It shows in a visual manner runtime scenarios of the system as a user goes through the various stages of deploying a running dashboard.

The diagram below depicts the deployment of the various components of system in order to regenerate an already developed dashboard. It shows the flow of events through the components, the communication flow which in the show the results to the user.

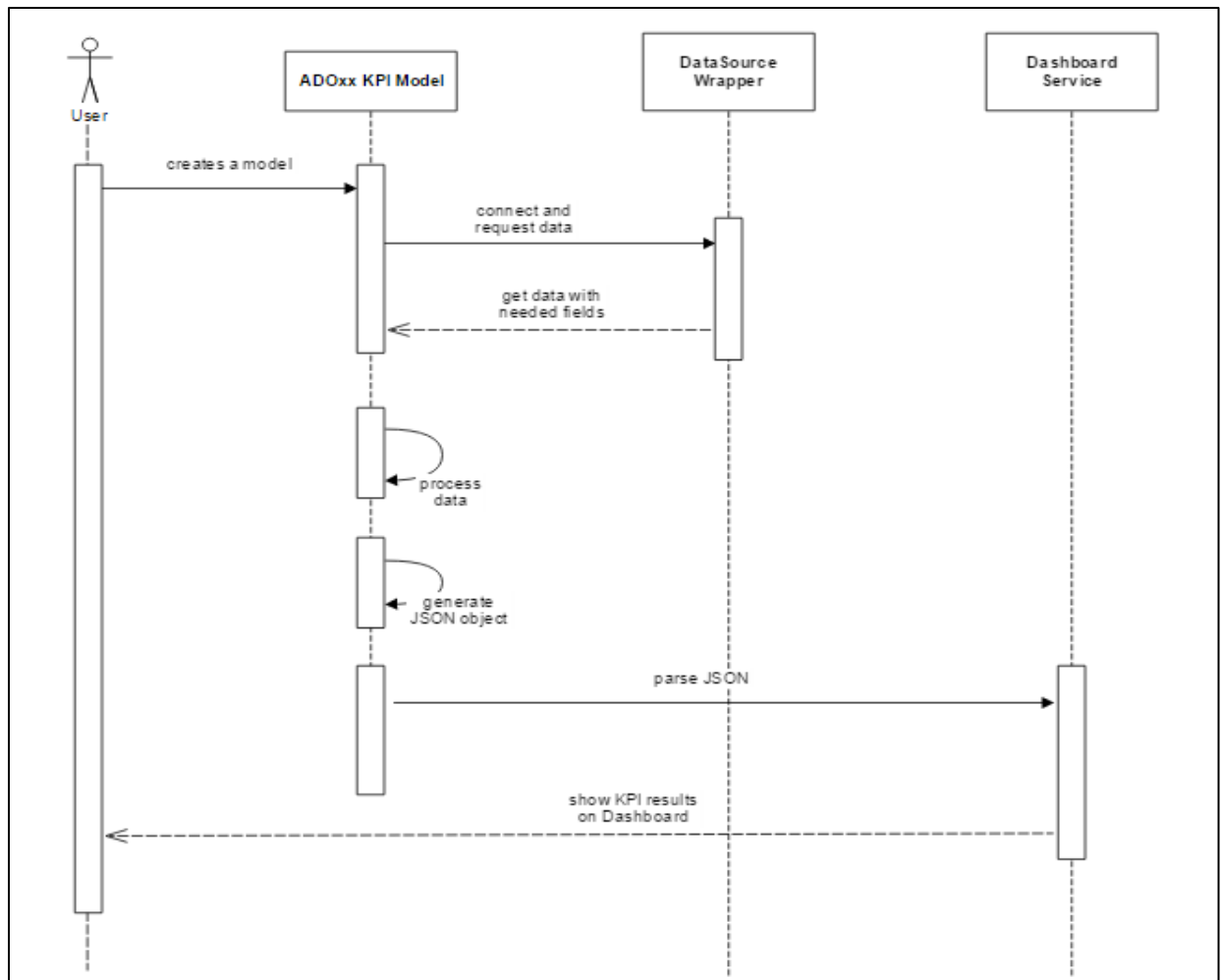


Figure 2. 3 KPI Dashboard Sequence Diagram

3. Development of the KPI Dashboard Components

This document pays particular focus on two (2) main components which are the DataSource Wrapper and the KPI Model components. The DataSource Wrapper is a Java component which is exposed as service that allows several data source modules to be created and added to it. Far different from that, the KPI Model is created based on ADOxx meta-modeling platform. This platform provides to the user the possibility to develop individual and domain-specific graphical modeling language, by developing your syntax, semantic and graphical notation for your modeling concepts.

3.1. The DataSource Wrapper Overview

The DataSource wrapper is a Java component that contains several modules used to get data from the different type of data source. It contains implementation modules for various data sources which are exposed through a java interface.

The general idea of the DataSource Wrapper is that, it gets as input the type of the data source and its configuration in a JSON format a provided by the JSON object generated from the KPI model component. The input is recognized by the module and returns a specific JSON with the value returned by the service. Each data source module should implement the interface which contains methods that will return the module type information and brief documentation describing the function of each method in the java interface.

3.1.1. Dashboard Maven project for implementing the DataSource Wrapper

The Dashboard maven project source code for the deployment of the DataSource Wrapper is provided in the ADOxx Web Dashboard github repository as an open source project.

(See this link to download: “<https://github.com/ADOxx-org/KpiDashboard-Web>”). Download and import the project source code into preferably Eclipse IDE as an existing project.

3.1.1.1. Java Packages and constituent classes in the Dashboard project

The Dashboard project mainly contains five (5) packages in the “src/main/java” which has the basic required classes to be able to deploy this project on a server.

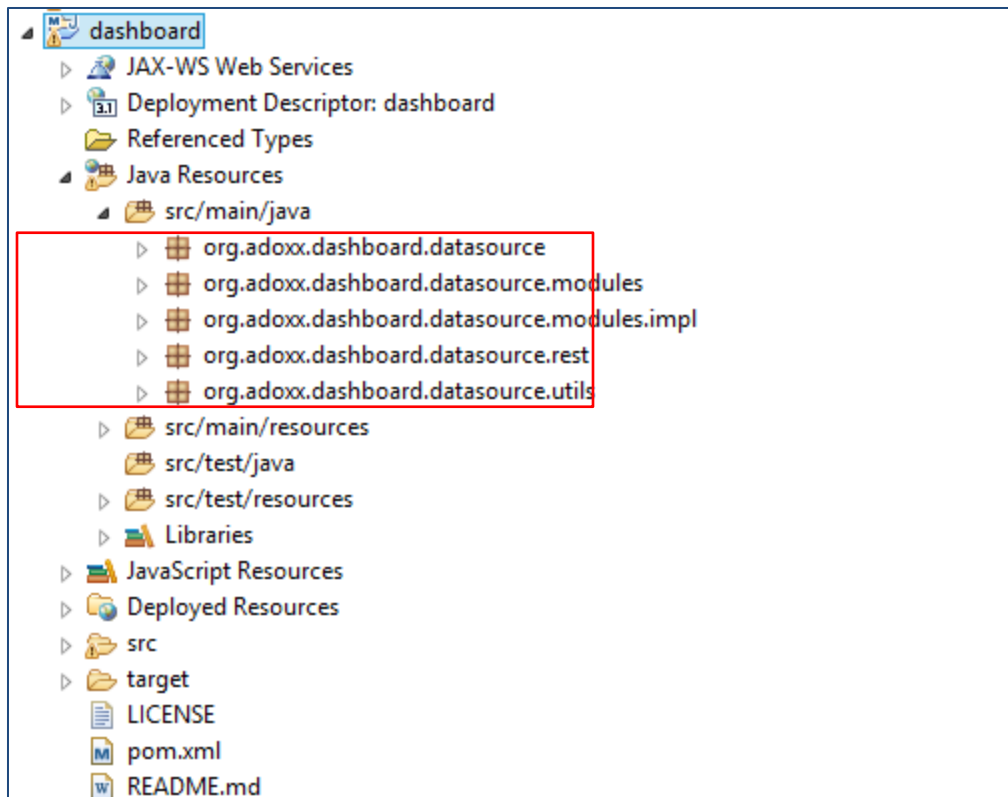


Figure 2. 4 Dashboard Project to be imported in Eclipse

- i. **Package org.adoxx.dashboard.datasource** : contains the “DSWrapper” class which gets all the information required from the available data source modules. Further details on the class are given later in the document.
- ii. **Package org.adoxx.dashboard.datasource.modules** : contains a java interface which provides several methods required by DataSource modules. There exists description of each method and its return value.

Interface’s abstract methods:

- **getUniqueName()**: returns the unique name of the component/datasource
- **getDescription()**: returns a JSON object containing a small description of the module in different languages
- **getConfigurationDescription()**: return the configuration JSON required by the module in order to work correctly. This JSON object must contain a JSON object for each parameter needed to be configured and each parameter must contain a JSON object for its description in different languages and a JSON string for passing the value of the parameter (that in this method will be empty).
- **obtainData()**: This method calls the managed service using the provided configuration and returns the service output in a structured way. The configuration JSON as returned by the “getConfigurationDescription” method,

but with the value field set. Returns a JSON object representing a table of data when possible, else a general output. In case of the table representation it must contain an array that describes the data columns and an array with a JSON object for each row.

- **isDataStructured()**: Returns true if the obtainData method returns a JSON representing a table, false when it returns a general output JSON

- iii. **Package org.adoxx.dashboard.datasource.modules.impl** : this is an implementation package where all the DataSource modules are configured. There is the possibility to extend by adding new modules. The module implements all the methods provided by the “DSModuleI” interface and transforms the return value of the methods in a unique way depending on the module. Available modules in the Dashboard project are DSMySQL, DSExcel, DSMSSQLServer, DSRESTService, DSTripleStore and DSJsonFile. Details of modules are described later in this document.
- iv. **Package org.adoxx.dashboard.datasource.rest** : this mainly provides a REST Service which implements the “DSWrapper” class as a Web service providing paths to resources to retrieve and manage some details of DataSource modules.
- v. **Package org.adoxx.dashboard.datasource.utils** : contains a class which provides some general functionality to the Web Service to upload a file to the server.

3.1.1.2. MySQL Module Implementation

This module is based on a MySQL database which will provide data to the wrapper; therefore it is required to have an installation of the database to use this module.

The DSMySQL module contains methods as described above.

- **getUniqueName()**: this method has no argument but returns a string “mysql-datasource”
- **getDescription()**: this method has no argument but returns a JSON object with description of the module which could be in different languages.
- **getConfigurationDescription()**: this method has no argument but returns the configuration JSON required by the module in order to work correctly.
For this module, the JSON object returned contains parameters and its descriptions required to connect to the MySQL database such as:
 - host
 - port
 - database
 - username
 - password
 - query

- **obtainData** (JsonObject configuration) : this method has the returned configuration object from the above method as an argument. It then uses the parameters provided by the configuration object to connect to the MySQL database and retrieves the selected fields. The final return value for this method is JSON object in the format shown below but with the corresponding values set.

```
{
  columns : ['value', 'instantTime', 'field3']
  data : [
    {
      value : '...',
      instantTime : '...',
      field3 : '...'
    },{
      value : '...',
      instantTime : '...',
      field3 : '...'
    },{
      value : '...',
      instantTime : '...',
      field3 : '...'
    }
  ]
}
```

- **isDataStructured()**: Returns “true” value since the “obtainData” method returned a JSON representing a table.

3.1.1.3. MSSQL Server Module Implementation

This module is based on a MSSQL server which will provide data to the wrapper; therefore it is required to have an installation of the database to use this module.

The DSMSSQLServer module contains methods as described above.

- **getUniqueName()**: this method has no argument but return a string “mssqlserver-datasource”
- **getDescription()**: this method has no argument but returns a JSON object with description of the module which could be in different languages.
- **getConfigurationDescription()**:this method has no argument but returns the configuration JSON required by the module in order to work correctly. For this module, the JSON object contains parameters and its descriptions required to connect to the MSSQL server such as:
 - host
 - port
 - database

- username
- password
- query
- **obtainData** (JsonObject configuration) : this method has the returned configuration object from the above method as an argument. It then uses the parameters provided by the configuration object to connect to the MSSQL server and retrieves the selected fields. The final return value for this method is JSON object in the format shown below but with the corresponding values set.


```
{
  columns : ['value', 'instantTime', 'field3']
  data : [
    {
      value : '...',
      instantTime : '...',
      field3 : '...'
    }, {
      value : '...',
      instantTime : '...',
      field3 : '...'
    }, {
      value : '...',
      instantTime : '...',
      field3 : '...'
    }
  ]
}
```
- **isDataStructured()**: Returns “**true**” value since the “obtainData” method returned a JSON representing a table.

3.1.1.4. Excel Module Implementation

This module is based on Microsoft Excel sheets which will provide data to the wrapper; therefore it is required to have an installation of the database to use this module.

The DSEExcel module contains methods as described above.

- **getUniqueName()**: this method has no argument but return a string “**excel-datasource**”
- **getDescription()**: this method has no argument but returns a JSON object with description of the module which could be in different languages.
- **getConfigurationDescription()**: this method has no argument but returns the configuration JSON required by the module in order to work correctly. For this module, the JSON object contains parameters and its descriptions required to connect to the particular Excel sheet such as:
 - filePath

- sheetNumber
- password (OPTIONAL)
- cellSeries (i.e. a comma “,” separated list or rage of columns/rows that represent the set of cells where the series of data are described)
- cellValues (i.e. a comma “,” separated list or rage of columns/rows that represent the set of cells that contain the data of the series)
- query
- **obtainData** (JsonObject configuration) : this method has the returned configuration object from the above method as an argument. It then uses the parameters provided by the configuration object to connect to the Excel sheet and retrieves the selected fields. The final return value for this method is JSON object in the format shown below but with the corresponding values set.


```
{
  columns : ['value', 'instantTime', 'field3']
  data : [
    {
      value : '...',
      instantTime : '...',
      field3 : '...'
    }, {
      value : '...',
      instantTime : '...',
      field3 : '...'
    }
  ]
}
```
- **isDataStructured**(): Returns “true” value since the “obtainData” method returned a JSON representing a table.

3.1.1.5. REST Service Module Implementation

This module is based on REST Service which will provide data to the wrapper; therefore it is required to have an installation of the database to use this module. The DSRESTService module contains methods as described above.

- **getUniqueName**(): this method has no argument but return a string “rest-datasource”
- **getDescription**(): this method has no argument but returns a JSON object with description of the module which could be in different languages.
- **getConfigurationDescription**(): this method has no argument but returns the configuration JSON required by the module in order to work correctly. For this module, the JSON object contains parameters and its descriptions required to connect to the REST service such as:
 - endpoint

- method
- requestContentType
- querystring
- postData
- additionalHeaders
- **obtainData** (JsonObject configuration) : this method has the returned configuration object from the above method as an argument. It then uses the parameters provided by the configuration object to connect to the REST Service and retrieves the selected fields. The final return value for this method is JSON object in the format shown below but with the corresponding values set.


```

      {
        dataFormat : 'application/xml',
        data : '...'
      }
      
```
- **isDataStructured()**: Returns “**false**” value since the “obtainData” method returned a general output instead of JSON representing a table

3.1.1.6. TripleStore Module Implementation

This module is based on TripleStore which will provide data to the wrapper; therefore it is required to have an installation of the database to use this module.

The DSTripleStore module contains methods as described above.

- **getUniqueName()**: this method has no argument but return a string “**triplestore-datasource**”
- **getDescription()**: this method has no argument but returns a JSON object with description of the module which could be in different languages.
- **getConfigurationDescription()**: this method has no argument but returns the configuration JSON required by the module in order to work correctly. For this module, the JSON object contains parameters and its descriptions required to connect to the TripleStore such as:
 - endpoint
 - sparqlQuery
- **obtainData** (JsonObject configuration) : this method has the returned configuration object from the above method as an argument. It then uses the parameters provided by the configuration object to connect to the TripleStore and retrieves the selected fields. The final return value for this method is JSON object in the format shown below but with the corresponding values set.

```

{

```

```

columns : ['value', 'instantTime', 'field3']
data : [
  {
    value : '...',
    instantTime : '...',
    field3 : '...'
  }, {
    value : '...',
    instantTime : '...',
    field3 : '...'
  }
]
}

```

- **isDataStructured()**: Returns “**true**” value since the “obtainData” method returned a JSON representing a table.

3.1.1.7. JsonFile Module Implementation

This module is based on JsonFile which will provide data to the wrapper; therefore it is required to have an installation of the database to use this module. The DSJsonFile module contains methods as described above.

- **getUniqueName()**: this method has no argument but return a string “**json-datasource**”
- **getDescription()**: this method has no argument but returns a JSON object with description of the module which could be in different languages.
- **getConfigurationDescription()**: this method has no argument but returns the configuration JSON required by the module in order to work correctly. For this module, the JSON object contains parameters and its descriptions required to connect to the Json file such as:
 - path
 - content
- **obtainData** (JsonObject configuration) : this method has the returned configuration object from the above method as an argument. It then uses the parameters provided by the configuration object to connect to the JsonFile and retrieves the selected fields. The final return value for this method is JSON object in the format shown below but with the corresponding values set.


```

{
  dataFormat : 'application/xml',
  data : '...'
}

```
- **isDataStructured()**: Returns “**true**” value since the “obtainData” method returned a JSON representing a table.

3.1.2. Detailed Description of DataSource Wrapper Class

In the implementation code of the DSWrapper class, initially a static moduleList object is created with its return type as List with the DSModuleI interface as its argument as shown below:

```
▪ public static List<DSModuleI> moduleList = new ArrayList<DSModuleI>();
```

Subsequently, the DataSource modules created can be added to the above “moduleList” object as shown below.

```
[ moduleList.add(new DSJsonFile());  
  moduleList.add(new DSExcel());  
  moduleList.add(new DSMSSQLServer());  
  moduleList.add(new DSMySQL());  
  moduleList.add(new DSRESTService());  
  moduleList.add(new DSTripleStore()); ]
```

DSWrapper contains two (2) main java methods namely the “*getModules*” and the “*callModule*”. These methods perform some operation on the modules and return an output.

- *getModules()* : this method returns a JSON object containing the return values of all the methods present in each of the modules added to the *moduleList* above.
- *callModule(String moduleName, JsonObject moduleConfiguration)* : this method returns a JSON object and has the arguments of the name of the module and the module configuration object. It checks if the module configuration JSON object has its key/value pair set and subsequently get the actual data contained in the module specified.

3.1.3. Implementing the DataSource Wrapper as REST Web Service

The DataSource Wrapper is exposed through a web service with the “**RESTService**” class under the “**org.adoxx.dashboard.datasource.rest**”. To explain how it works, the DataSource Wrapper package is imported into the RESTService class in order to be able to call its methods. The RESTService is annotated with the path “datasourceWrapper” to be added to the URI for accessing the service. The web service provides two (2) relevant resources which are:

- i. **getModules()** : this method gets all the modules from the DSWrapper and returns the result as a string. It is annotated with a “**GET**” http request, a “**/getModules**” path and a “**Produces**” annotation to set MediaType to JSON.
- ii. **executeModule()** : this is POST method which takes the module name and the configuration as its parameters and calls the DSWrapper method which returns the data values from the specified module. The method is annotated with the path “**/executeModule**” and its “**Consumes**” and “**Produces**” annotations are of media type JSON.

3.2. The KPI Model

This component is based on the ADOxx.org platform which allows for the definition of a meta-model and the creation of an actual model to represent aspects of the KPI required by the dashboard. In defining the meta-model, classes and their relations are created in the ADOxx development platform as well as respective attributes created.

3.2.1. ADOxx Experimentation Library as a default library

In order to have a KPI modeling language, we need to create a meta-model on the ADOxx development environment. Here an installation of the ADOxx platform on your computer is required to get started.

(See this link for installation support: <https://www.adoxx.org/live/download-15>)

- Assuming installation is complete or you already have ADOxx existing on your computer, open the development toolkit and go to “Library Management”, “Libraries” then “Settings”.

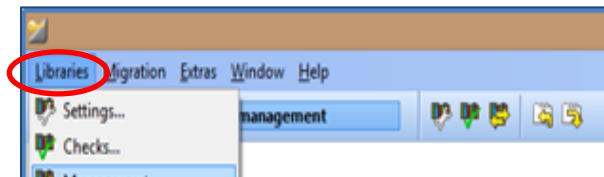


Figure 3. 1 Library Management settings

- At Library Management, select the dynamic library under the Experimentation library. Then click on the “Class Hierarchy” and add the meta-model view to see classes in detail.

3.2.2. Defining Classes and Attributes for the KPI modeling language

Now we define classes, attributes and relations for the KPI modeling language. Below are the details of the classes and associated definitions. (See this link for creating classes and attributes respectively: “https://www.adoxx.org/live/create_modelling_class” and “https://www.adoxx.org/live/create_string_attribute”).

Define classes and attributes below following the provided links:

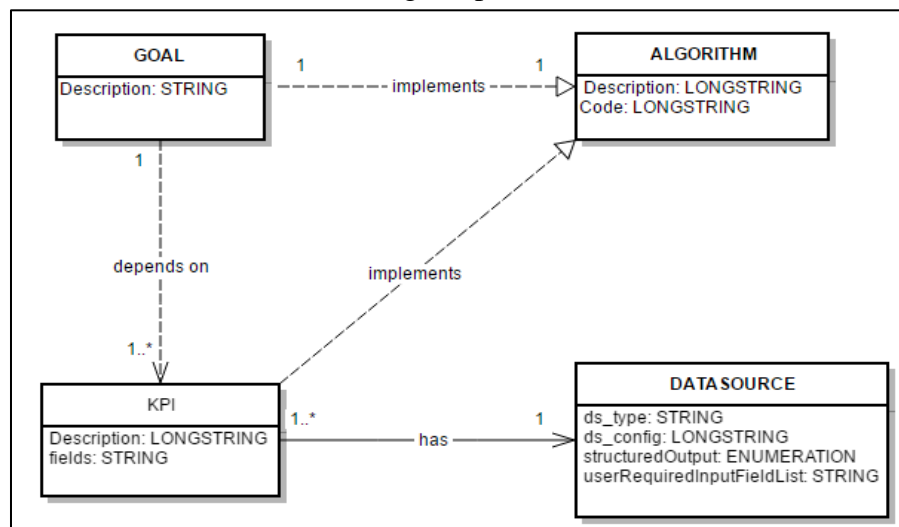


Figure 3. 2 Class Diagram of KPI Meta-Model

A. Classes with Attributes:

1. **Datasource**

- **ds_type** (attribute type: STRING)
- **ds_config** (attribute type: LONGSTRING)
- **structuredOutput** (attribute type: ENUMERATION)
- **userRequiredInputFieldList** (attribute type: STRING)

2. **KPI**

- **Description** (attribute type: LONGSTRING)
- **fields** (attribute type: STRING)

3. **Goal**

- **Description** (attribute type: LONGSTRING)

4. **Algorithm**

- **Description** (attribute type: LONGSTRING)
- **Code** (attribute type: LONGSTRING)

(See also this link for creating relation classes below:

[“https://www.adoxx.org/live/create_relation_class”](https://www.adoxx.org/live/create_relation_class))

B. Relations:

- **has_datasource** (KPI-to-Datasource)
- **depend_on_kpi** (KPI-to-KPI)
- **kpi_has_algorithm** (KPI-to-Algorithm)
- **evaluated_with_kpi** (Goal-to-KPI)
- **depend_on_goal** (Goal-to-Goal)
- **goal_has_algorithm** (Goal-to-Goal)

3.2.3. Defining Graphical Representation for Classes and Notebook Attributes

We now create graphical representations for each of the classes and relation classes above. (See this link: “https://www.adoxx.org/live/create_static_graphrep”). Give different graphical representation for each of the classes and relation classes. For more details on example graphReps go to: “<https://www.adoxx.org/live/adoxx-graphrep-repository-wiki/-/wiki/GRAPHREP+Repository/FrontPage>”.

For the classes and relation classes to visible we need to define “attrRep” attributes by add all the attributes created above to the Notebook. (Check this link to see attributes defined in the Notebook: “https://www.adoxx.org/live/define_attrrep_attribute”).

3.2.4. Creating a Modeltype

We create a modeltype in the “Library Attributes” and add all the classes to the modeltype. (See link on creation of modeltype, “https://www.adoxx.org/live/create_modeltype”) in this case use the name “KPI Model” and change the classes included to conform with the classes and relations created above.

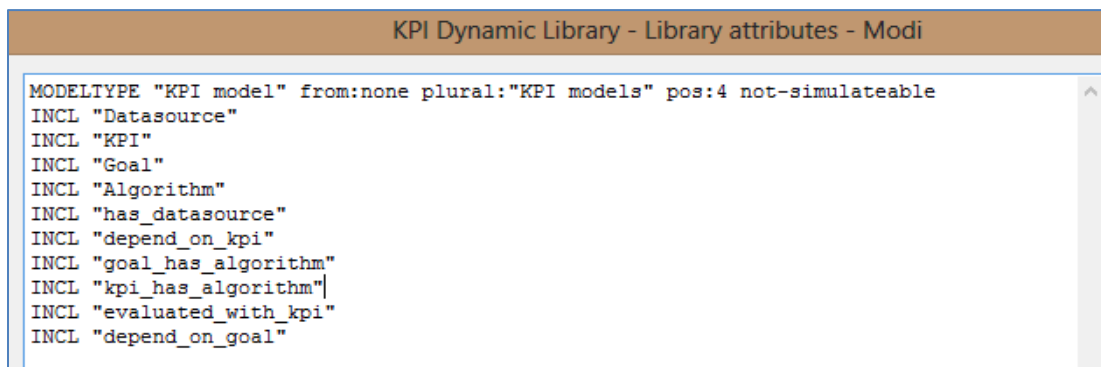


Figure 3. 3 Creating a Modeltype and adding Library attributes

3.2.5. Creating a User to access the Modeling Toolkit

A user is required to be created in connection with the corresponding library in order to be able to access the modeling toolkit. (See this link to create user: “https://www.adoxx.org/live/create_user”).

3.2.6. ADOxx Modeling Toolkit Set up

Open the modeling toolkit and enter the user credentials created above (i.e. User name and password) and click on “**Model**” at the top menu to create a new model (select the “KPI Model” modeltype). All classes should be checked to confirm the attributes were defined correctly in the Notebook.

A simple KPI model can be created as seen below:

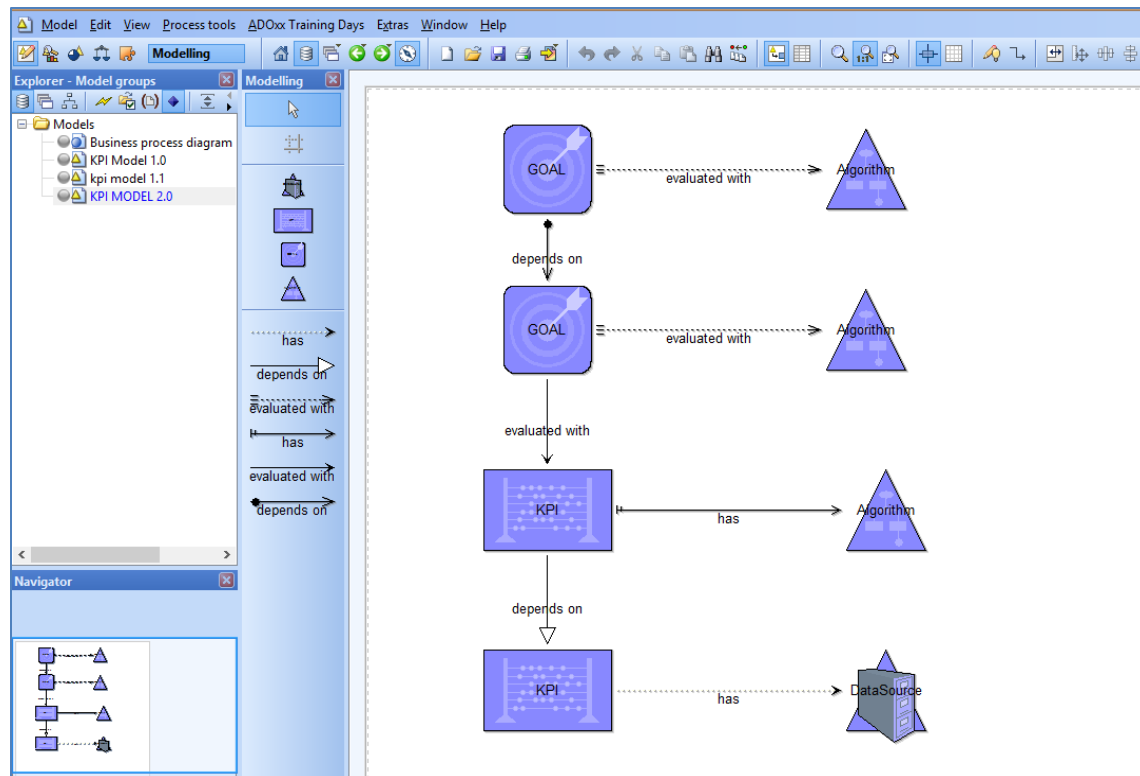


Figure 3. 4 Example of a KPI Model showing objects and their relations

Note: graphical representation of the Classes and relations could be different

3.2.7. AdoScript to retrieve Notebook Attribute values of the instances on a KPI model

The AdoScript language which adds functionality to models in ADOxx is used to retrieve all the required attributes values from each of the instances or objects in the model. The aim of this is to generate a JSON object in a format which required by the Web Dashboard service to work. The JSON is interpreted by the Dashboard thereby allowing widgets on the Dashboard to get the right values.

To bring this functionality to the model, a number of ADOxx Procedures (functions) are defined and subsequently called to perform a function. The script file which allows for the JSON to be generated can be accessed on the following github: “<https://github.com/ADOxx-org/KpiDashboard-MM/blob/master/src/scripts/KpiDashboardUtils.asc>”.

Copy the script into a text editor (e.g. Notepad++) and save the name of the file as “**KpiDashboardUtils**”. The script file contains two sets of “Procedures”, one is the set of functions (“10 Procedures in all”) that retrieves all the needed attribute values of each object on the KPI model and then transforms the output (into the Web Dashboard’s required JSON object) and finally export the result. The second is the set of functions (“3 Procedures in all”) which (it

helps the DataSource object to connect to the DataSource Wrapper component) will be discussed later.

3.2.8. Configuring “KpiDashboardUtils” in the ADOxx development toolkit

To be able to use the above script, import the file into development toolkit file management on the menu bar (i.e. “**Extras**” -> “**File Management**” under corresponding library).

Now go to the “**Library Management**”, select the required Library under the “**Settings**” tab and click on “**Library attributes**” -> “**Add ons**” -> “**External coupling**”. Here we create a script to execute the “**KpiDashboardUtils**” file when a button on the top menu bar is clicked.

Copy and Paste the script for execution below into the text area for the External coupling and click “**Apply**” then restart the modeling toolkit to view the button “**Generate Web Dashboard Json**” under the menu “**Extras**”:

```
#####  
ON_EVENT "AppInitialized"  
{  
    EXECUTE file:("db:\\KpiDashboardUtils.asc")  
}  
ITEM "Generate Web Dashboard Json" acquisition:"Extras" modeling:"Extras" analysis:"Extras"  
simulation:"Extras" evaluation:"Extras" importexport:"Extras"  
EXPORT_KPI_MODEL_IN_JSON  
#####
```

3.2.9. The Dashboard Helper

This is a Java Maven project which aids the configuration of the DataSource and KPI objects in the model. Its main function is to create a dialog boxes in order to set some parameters required to connect to a particular data source or select fields the data provided by the data source in the case of the KPI object.

There is a main class which executes each of the methods when the program is run. Available source codes for the Dashboard Helper are provided here in the ADOxx Web Dashboard github repository: “https://github.com/ADOxx-org/KpiDashboard-MM/tree/master/src/additional_files/dashboard-helper_src”.

3.2.9.1. Methods in the Dashboard Helper

The Dashboard Helper contains a number of methods which is used to create JDialog boxes to set parameters as mentioned earlier. All methods available to the helper read the final JSON generated as a result of the different available modules of the data sources configured for the dashboard. The content of this JSON is key/value pair of each of the required parameters for the

respective data sources present (i.e. key/value pair of parameters for Excel, MSSQLServer, MySQL and other data source not discussed in this document). The methods for the helper are:

- **chooseDataSourceType()** : This method gets all the names and descriptions of the data sources and presents it in a JDialog with the list of data source in a drop-down menu to be selected by the user.

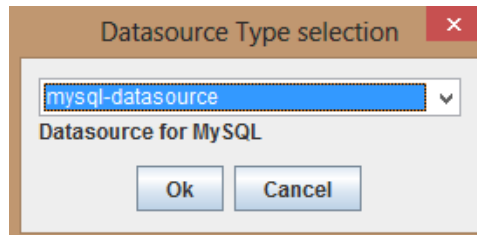


Figure 3. 5 Selecting a DataSource Type using the Dashboard Helper

- **chooseDataSourceConfiguration()** : this method retrieves the key “configuration” with an object as its value for the any selected data source from the above method. It further retrieves the all keys for the returned object which are the configuration parameters used to connect to the particular data source. Then the dialog box to input the values into parameters is called.

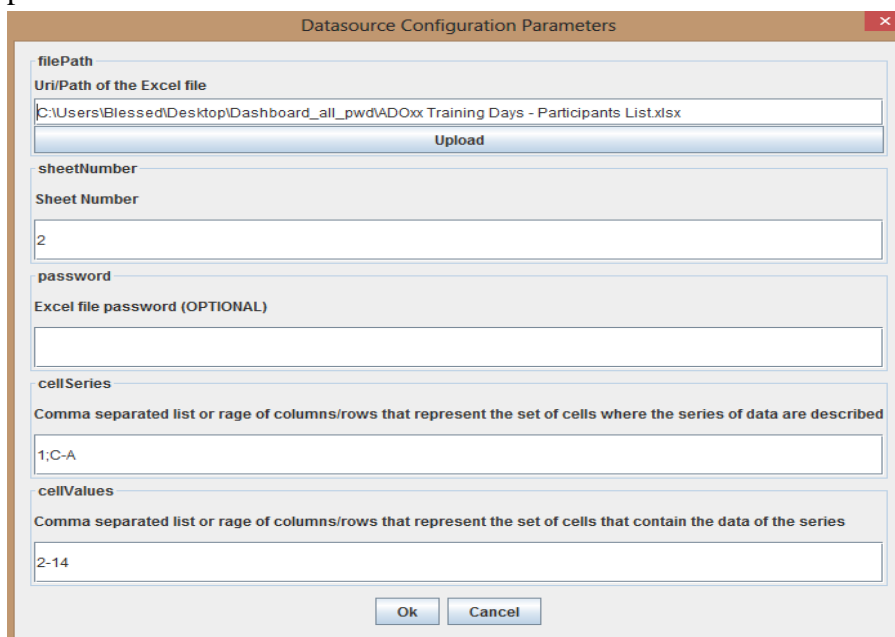


Figure 3. 6 Setting Configuration parameters for the selected DataSource type

- **chooseDataSourceUserInputs()** : this method simply creates a JDialog box to allow any user input values to set if there exist. The user can enter a “key” (for instance, a password) and a “description” of the key. This is required for instance, if an excel sheet is password-protected and it will subsequently be configured on the Web Dashboard to be entered any time data from the excel sheet accessed.

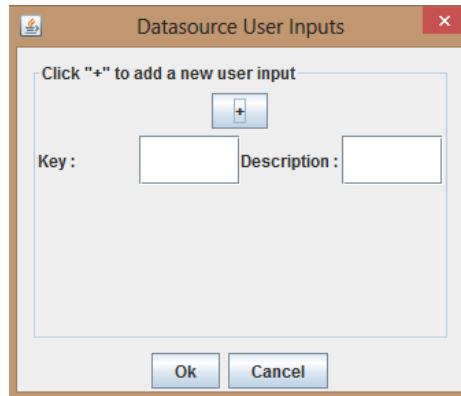


Figure 3. 7 Adding User Input

- **chooseKPIFields()** : this method is related to the KPI object in the KPI model and allows fields from the provided data by the data source to be specified and retrieved.

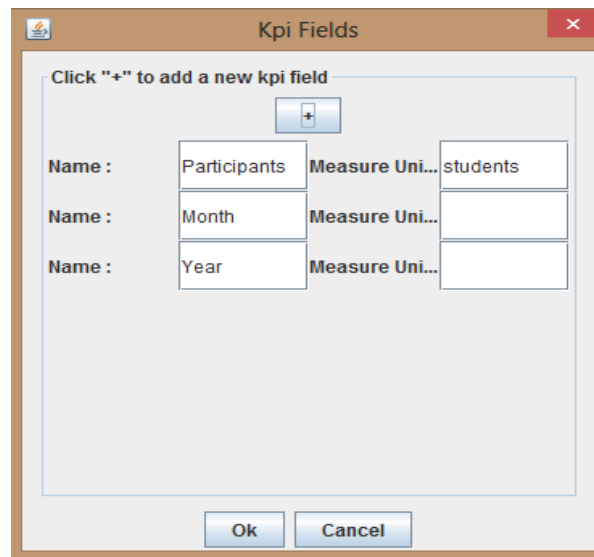


Figure 3. 8 Adding KPI Fields to be retrieved

3.2.9.2. Configuring the Dashboard Helper with the KPI Model

Create additional attributes to the DataSource and KPI classes as shown in 3.2.2 above.

Attributes to add are:

Class	Attribute	Type
1. DataSource	call_datasource_helper	PROGRAMCALL
2. KPI	call_dashboard_KPIFields	PROGRAMCALL

a) In creating the attribute “**call_datasource_helper**” for the DataSource class, select Datasource and create new attribute as shown below and click “Edit”.a

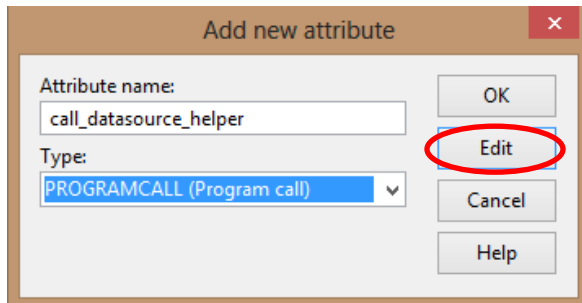


Figure 3. 9 Create New attribute for DataSource class

b) Set up the “**call_datasource_helper**” attribute as shown below:

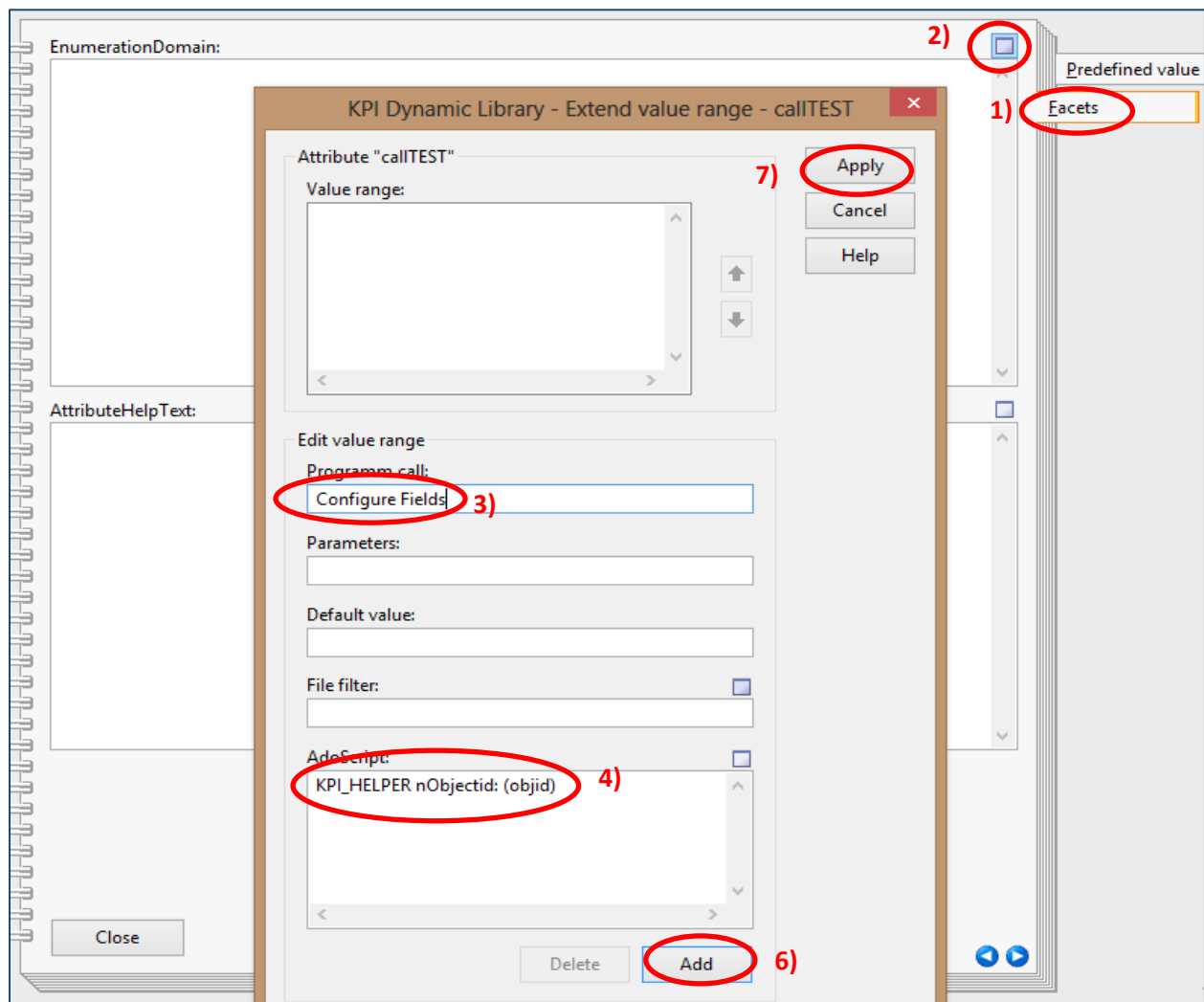


Figure 3. 10 Setting a program call to procedure to execute the Dashboard Helper jar

c) Add the new attribute to the attrRep Notebook setting it as a button as shown below:

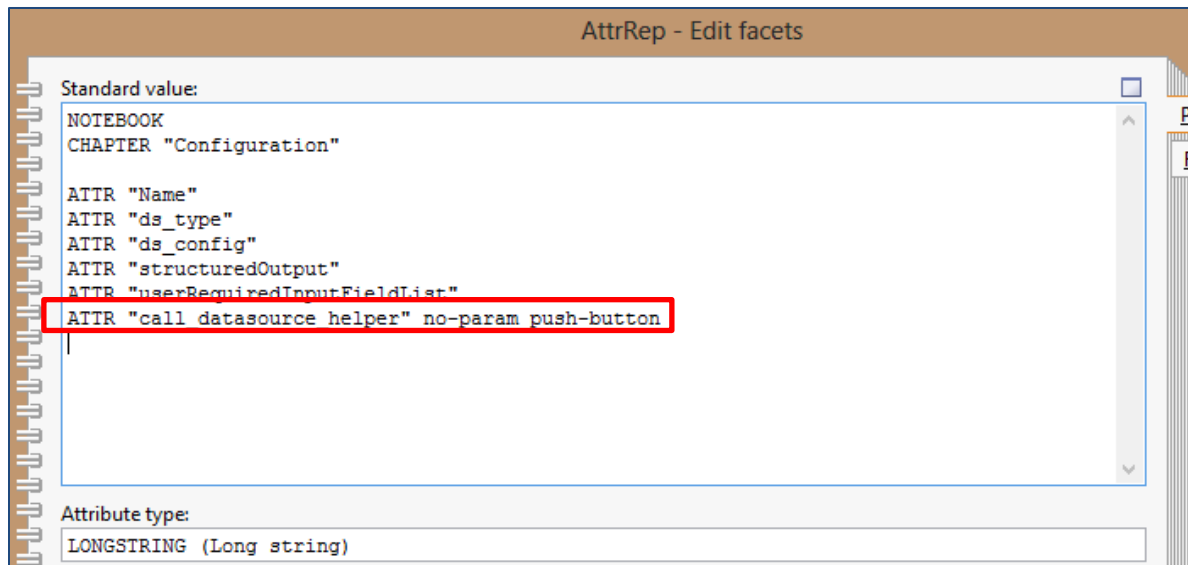


Figure 3. 11 Adding the new attribute to the DataSource “attrRep”

d) In the modeling toolkit, the datasource object now has the attribute showing and executes the dashboard_helper jar when clicked.

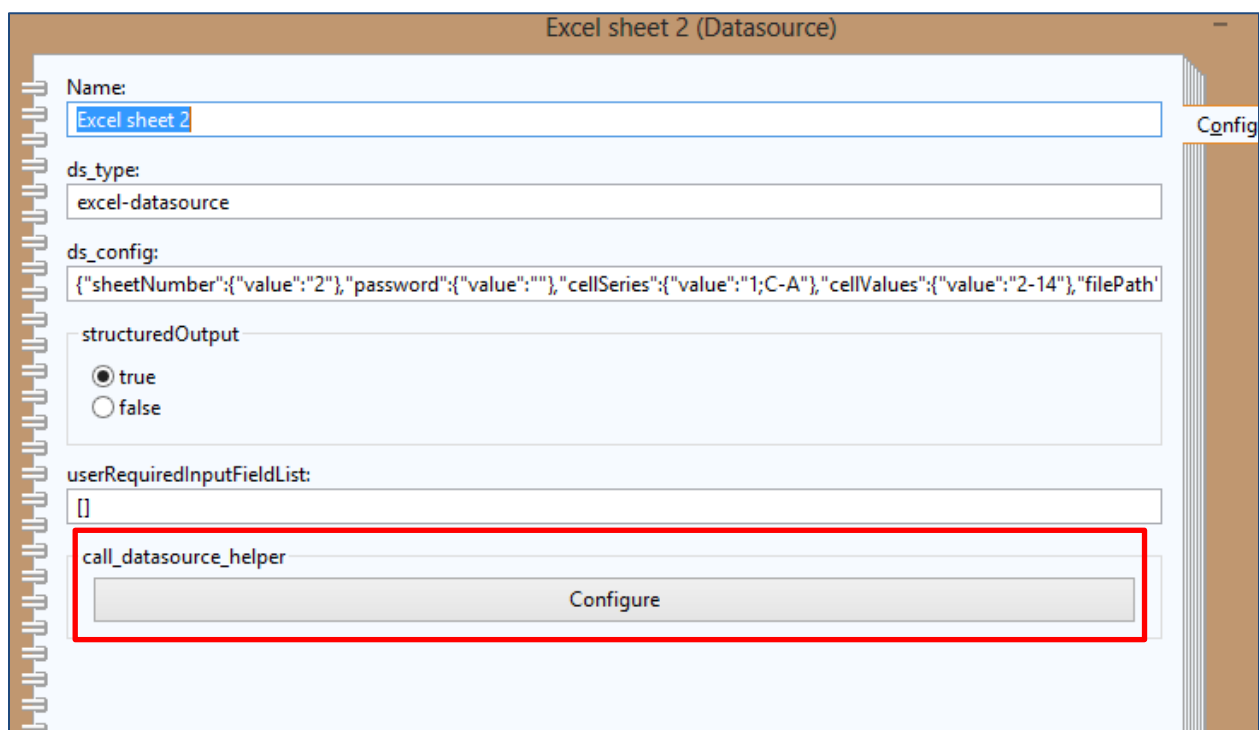


Figure 3. 12 Button to execute Dashboard helper jar

- Repeat steps a) to c) above to create the second attribute “**call_dashboard_KPIFields**” for the KPI class and set it up with the helper ADOxx procedure “KPI_HELPER” as depicted in the diagram.

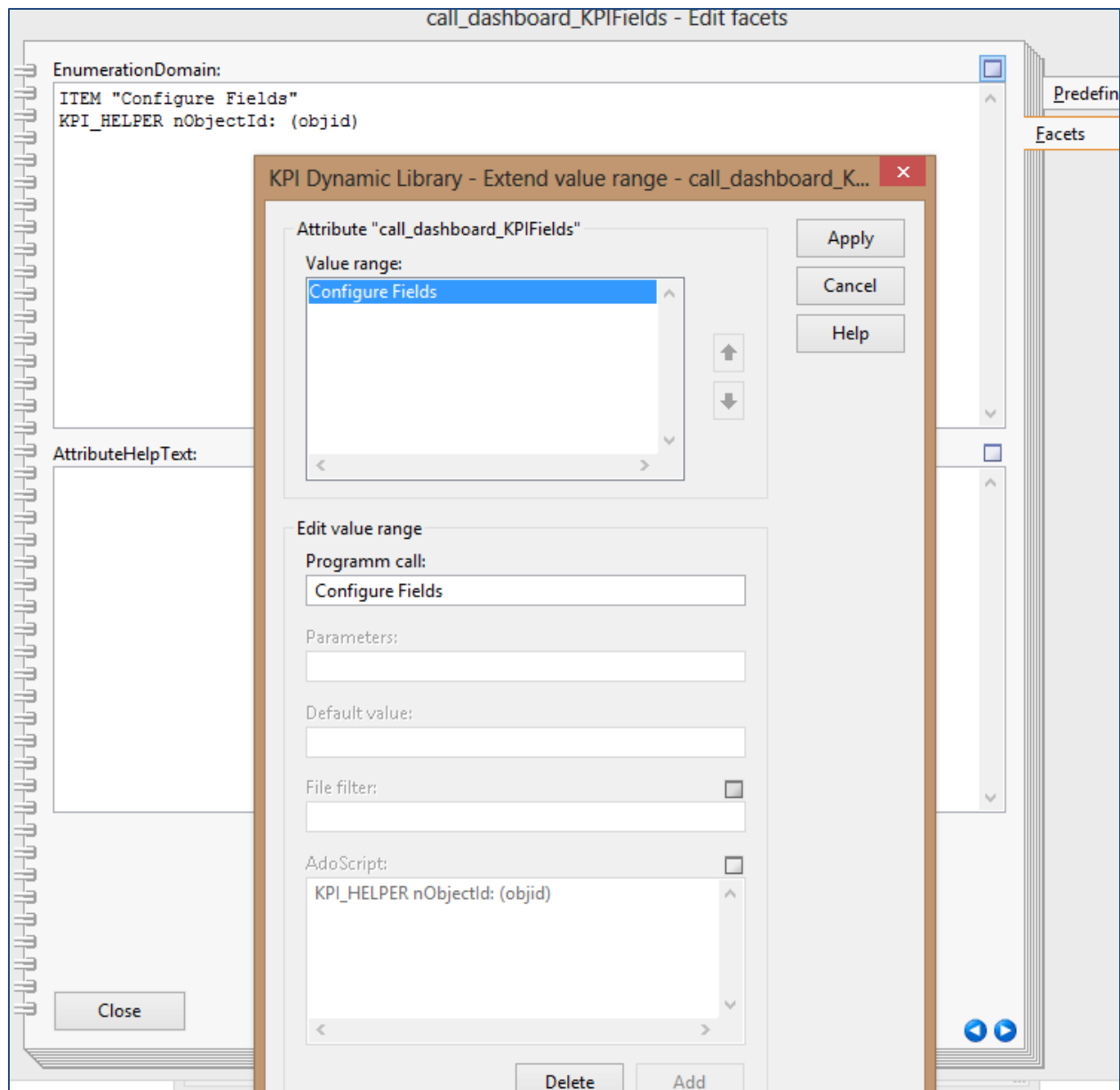


Figure 3. 13 Creating KPI attribute and setting program call to execute Helper jar

- The “Configure Fields” button allows for the “Fields” attribute to be set by triggering the Dashboard Helper.

Average yearly attendance (KPI)

Name:
Average yearly attendance

Description:
Is the average attendance over all years

Fields:
[{"name": "Average", "measureUnit": "students"}]

call_dashboard_KPIFields

Configure Fields

Description

Figure 3. 14 Button to configure KPI Fields with the Dashboard Helper

4. ADOxx Training Sessions Dashboard Example

Here, we try to explain how the dashboard can be set up and deployed using an example of an excel datasource with a sample ADOxx Training session data in order to visualize the data on the dashboard using a few widgets.

4.1. Prerequisite Tools and source codes

- i. Eclipse IDE with a web application server installed (e.g. JBoss Tomcat web server)
- ii. ADOxx Development and Modeling environments
- iii. Github repository source codes for this example Dashboard implementation provided at: [“https://github.com/ADOxx-org/KpiDashboard-MM”](https://github.com/ADOxx-org/KpiDashboard-MM)

4.2. Setting Up The Training Session Web Dashboard

4.2.1. ADOxx Environment Implementations

On the ADOxx Development Platform:

1. Import KPI library into the ADOxx development environment –
(***“Library Management” -> “Libraries” -> “Management” -> “Click Import” -> “browse to library file location of the KPI Library”***). Download library at:
[“https://github.com/ADOxx-org/KpiDashboard-MM/blob/master/KPIDashboardLibrary.abl”](https://github.com/ADOxx-org/KpiDashboard-MM/blob/master/KPIDashboardLibrary.abl)

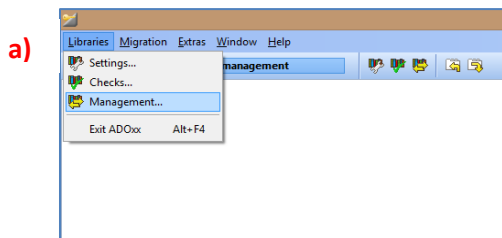


Figure 4. 1 ADOxx Library Management

b)

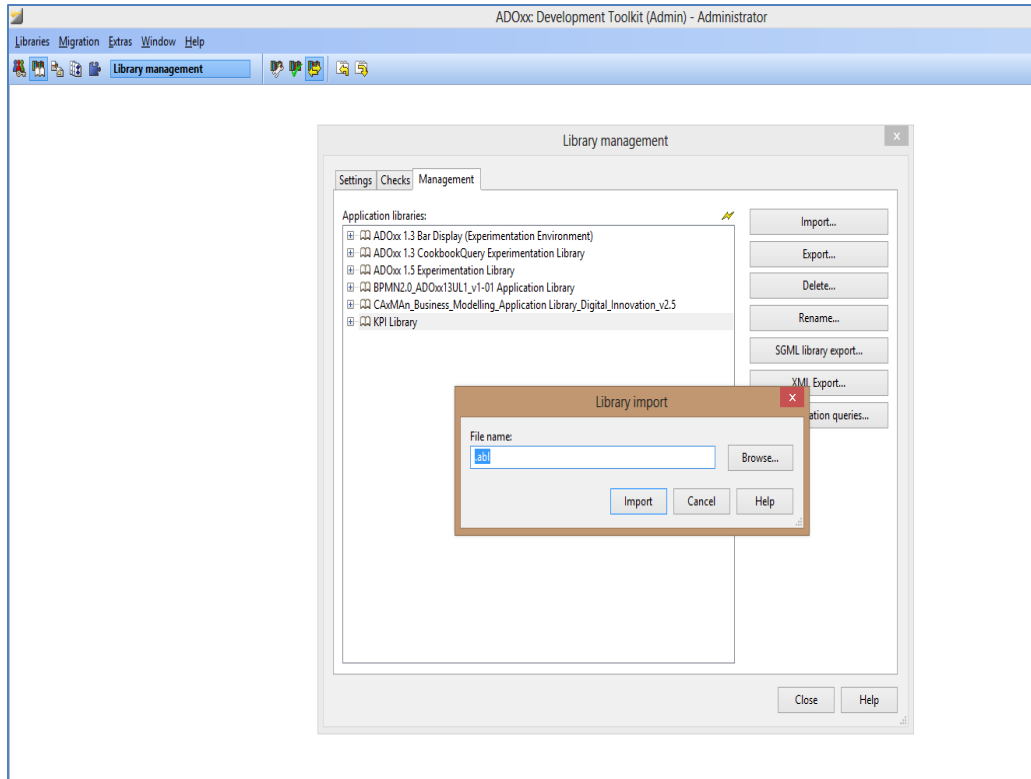
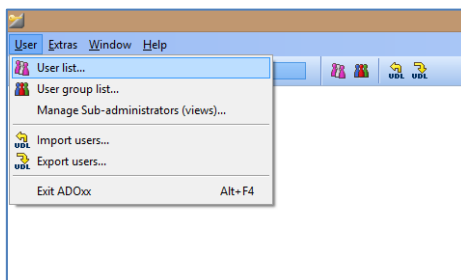


Figure 4. 2 KPI Dashboard Library Import

2. Create new user for the imported library –
(Go to “User Management” -> “User List” -> “Add” -> enter Username and password -> Click “User group” -> Double click “ADOxx” -> “OK”)

a)



b)

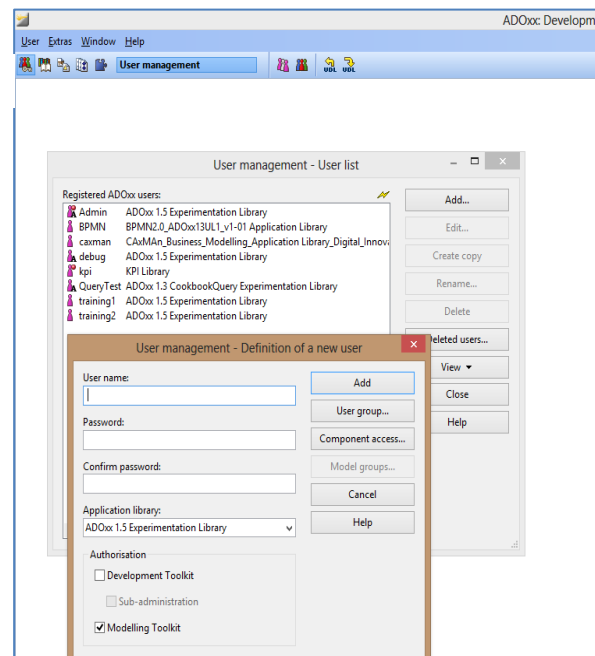


Figure 4. 3 Creating a New User and associate with the corresponding Library

- Import into the file management the AdoScript file “KPIDashboardUtils.asc” (i.e. script to convert the KPI into a JSON representation required Web interface) –
(Go to “Extras” -> “File Management” -> “Import” -> browse to file location and import)

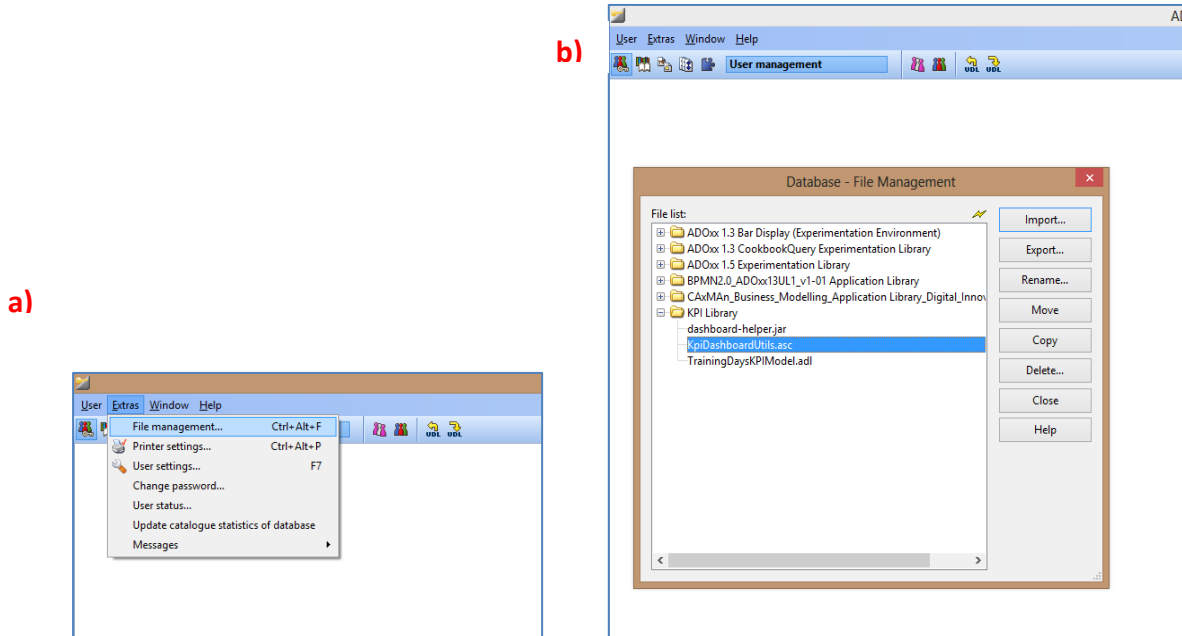


Figure 4. 4 File management for adding files to ADOxx database

- Copy the following AdoScript code (which helps to execute script which contains procedures) into the library attributes 'Add ons': -
(Go to “Library Management” -> “Settings” -> select the library -> click “Library attributes” -> then click “Add-ons”)

```
#####
ON_EVENT "AppInitalized"
{
    EXECUTE file:("db:\\KpiDashboardUtils.asc")
}
ITEM "Generate Web Dashboard Json" acquisition:"Extras" modeling:"Extras" analysis:"Extras"
simulation:"Extras" evaluation:"Extras" importexport:"Extras"
EXPORT_KPI_MODEL_IN_JSON
#####
```

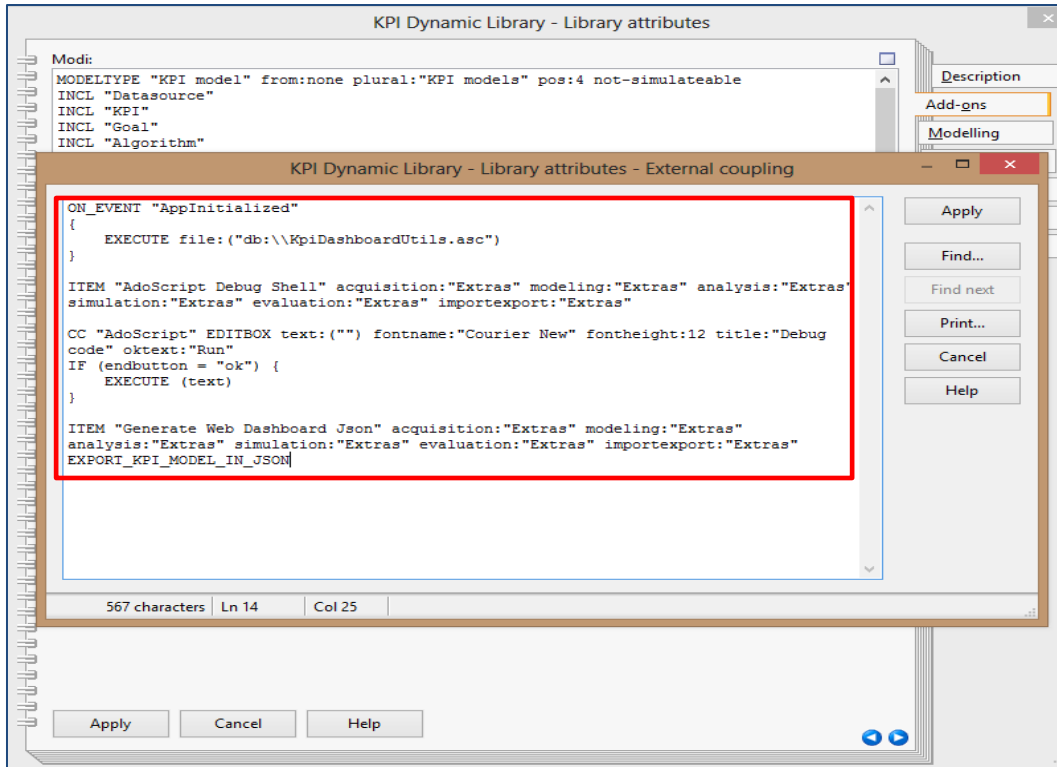


Figure 4. 5 ADOxx External coupling to execute AdoScript file

On the ADOxx Modeling Toolkit:

5. Open the modeling environment the new user credentials
6. Import the "TrainingDaysKPIModel.adl" file provided (this contains a predefined KPI model) – (*Go to "Import/Export" -> "Model" -> "ADL Import" -> "Models/Attribute profile"*)

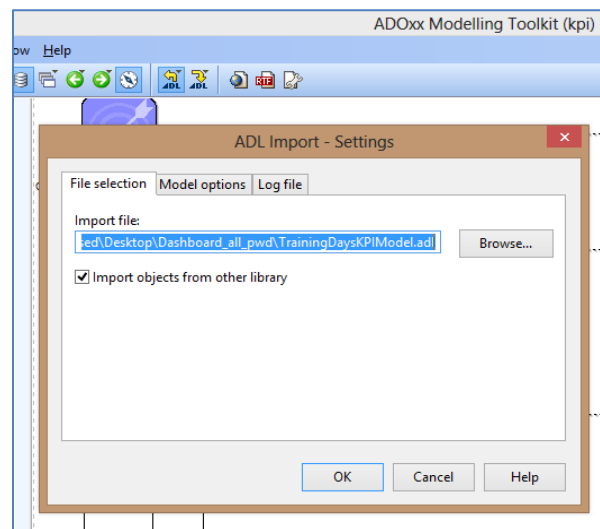


Figure 4. 6 KPI Model Import

4.2.1.1. ADOxx Training Session KPI Model

The “**TrainingDays-KPI**” model comprises of two distinct KPI models on one modeling area. Below are the screenshots of the models:

- 2 Goals, 3 KPI, 1 Datasource and 4 Algorithm objects for excel sheet 2

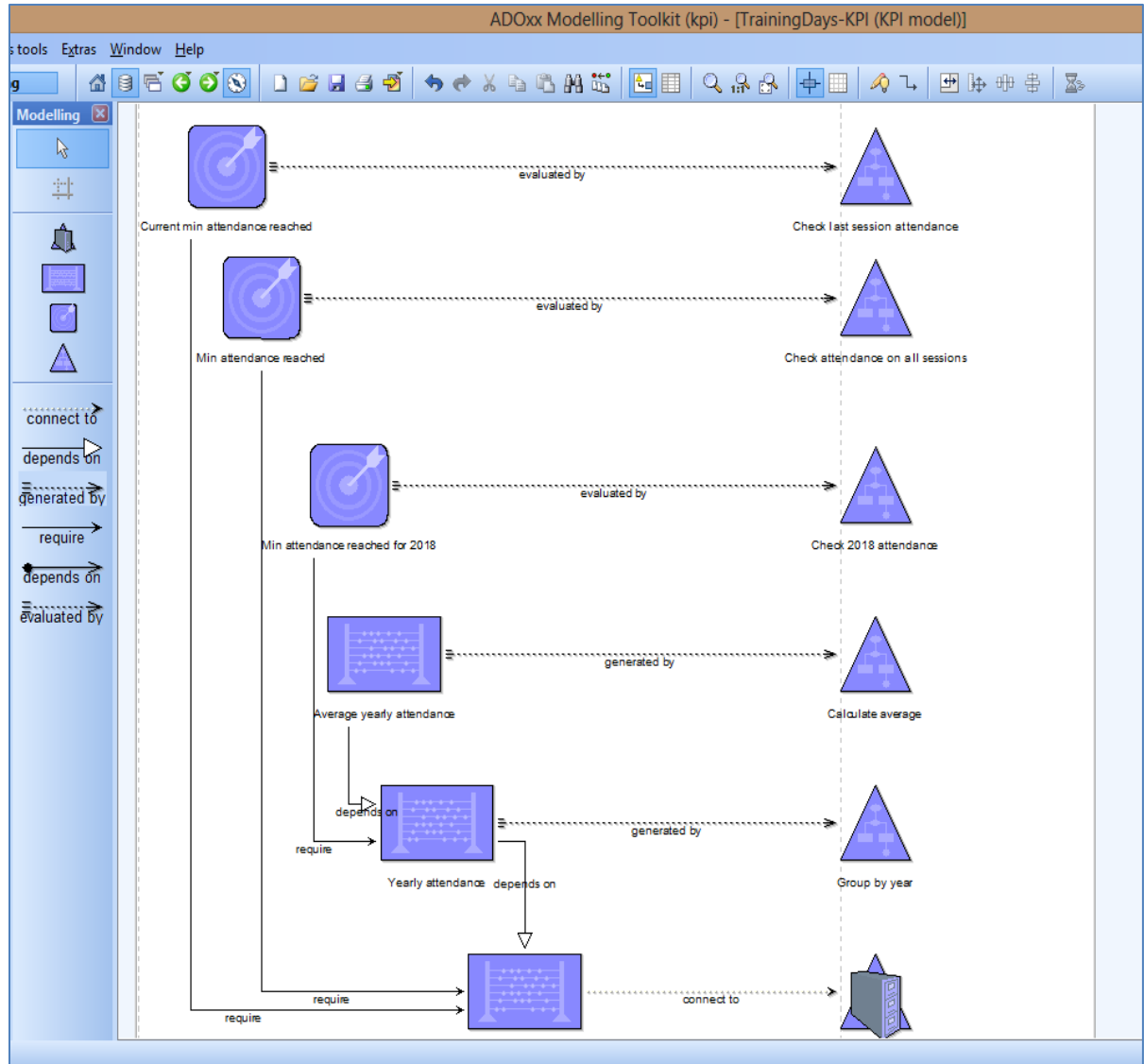


Figure 4. 7 KPI Model for the second Excel sheet available in the DataSource

- 1 Goals, 2 KPI, 1 Datasource and 3 Algorithm objects for Excel sheet 1

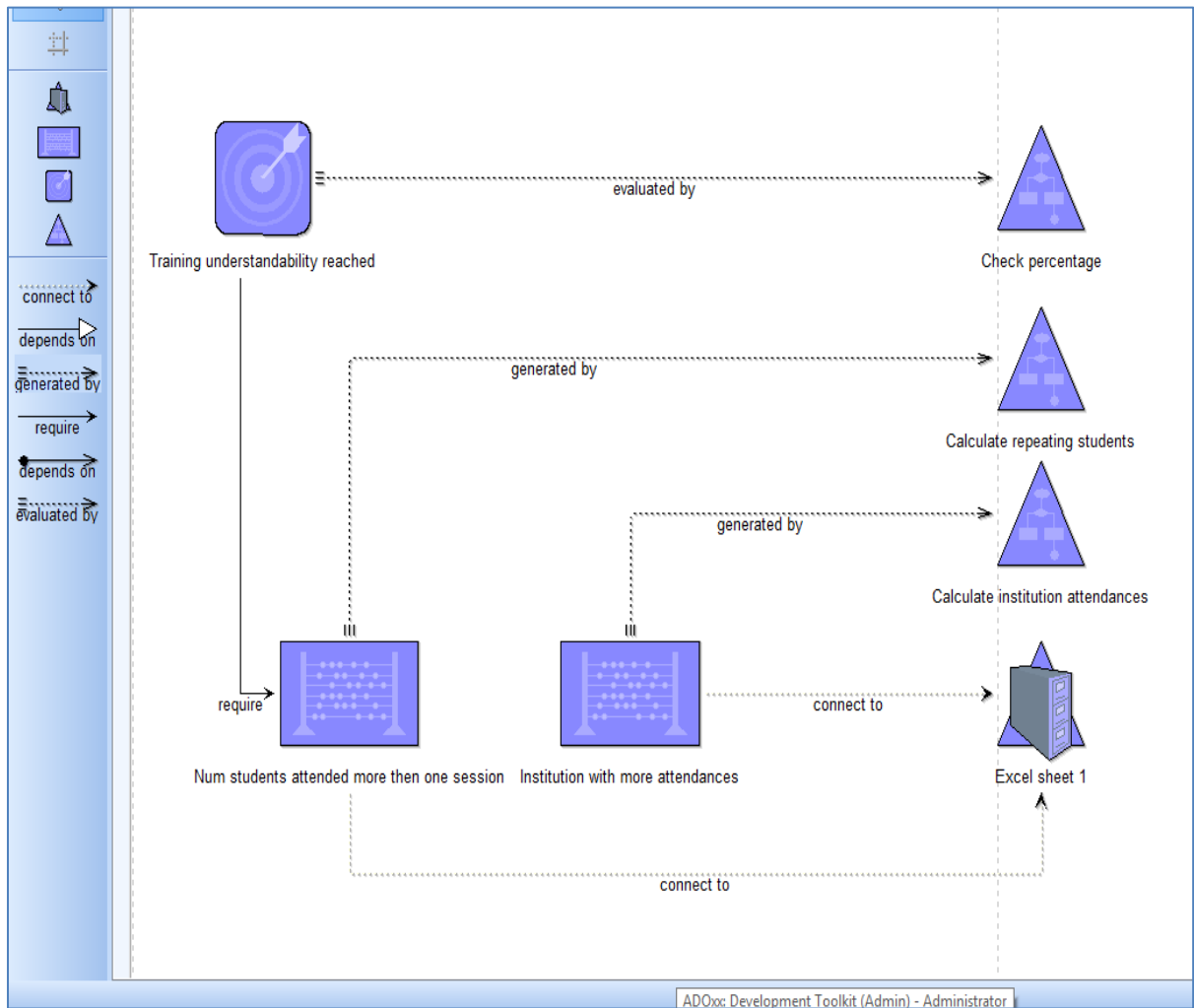


Figure 4. 8 KPI Model for the first Excel sheet available in the DataSource

4.3. Description of objects and relevant attribute values of the Training Session KPI model

The KPI model evaluates some data from the participants of the training usually organized by the ADOxx.org team. The data is Excel sheet provided by the DataSource wrapper with information about number of participants per training in a particular year and details of each participant.

In the first part the model consists of 2 Goals, 3 KPI, 1 Datasource and 4 Algorithm objects which have Notebook attributes of the classes.

Let's now review the attributes set for each one of the instances of the classes.

a) DataSource "Excel sheet 2"

i. *Double click on the object representation*

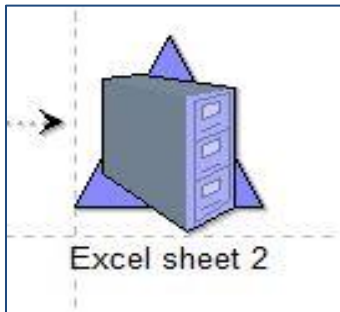


Figure 4. 9 DataSource object configured to select sheet 2 of the Excel data

ii. *Click on "Configure Fields" to call the java helper class which allows the attribute values to be set*

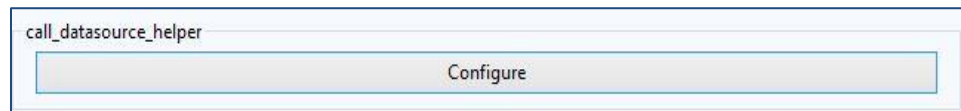


Figure 4. 10 Trigger the Dashboard Helper to configure DataSource object

iii. *"Dashboard REST Endpoint" is set with "<http://127.0.0.1:8080/dashboard/rest>" -> Click "Ok"*



Figure 4. 11 Setting Dashboard REST Endpoint

iv. *Select "excel-datasource" -> Click "Ok"*

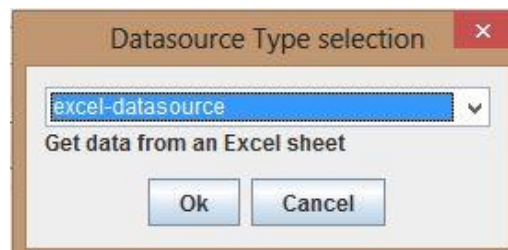


Figure 4. 12 Selecting a DataSource type

- v. *Configuration parameters to connect and extract data from excel sheet are set as shown below -> Click “Ok”*

Datasource Configuration Parameters

filePath
Uri/Path of the Excel file
C:\Users\Blessed\Desktop\Dashboard_all_pwd\ADOxx Training Days - Participants List.xlsx
Upload

sheetNumber
Sheet Number
2

password
Excel file password (OPTIONAL)

cellSeries
Comma separated list or rage of columns/rows that represent the set of cells where the series of data are described
1;C-A

cellValues
Comma separated list or rage of columns/rows that represent the set of cells that contain the data of the series
2-14

Ok Cancel

Figure 4. 13 Configurations Parameters required for connecting and retrieving data

- vi. *User Inputs (OPTIONAL); for example, excel password or user email, etc. -> Click “Ok” to finish*

Datasource User Inputs

Click "+" to add a new user input

+

Ok Cancel

Figure 4. 14 Setting User Inputs (OPTIONAL)

Note: Datasource attributes will be set as shown below:

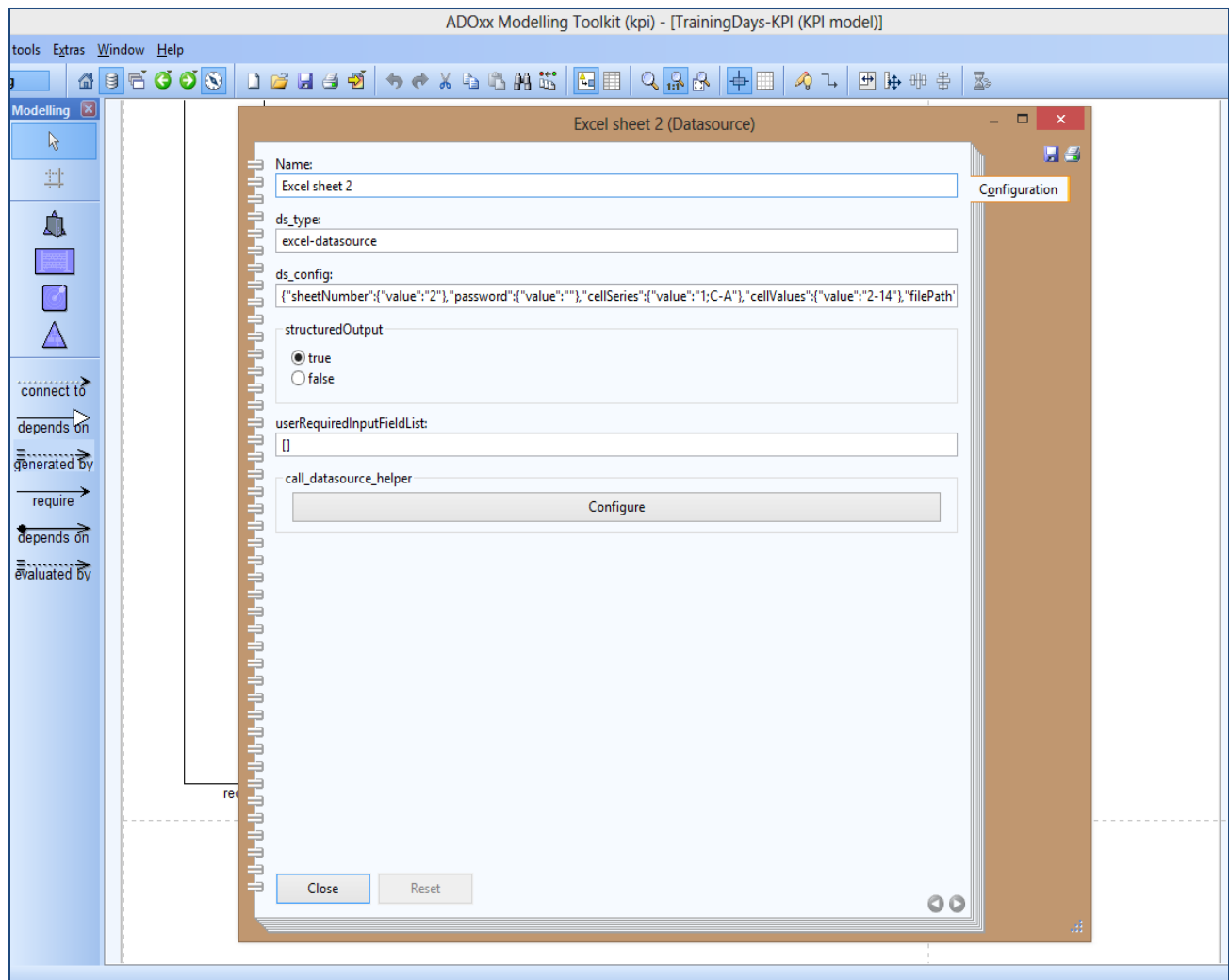
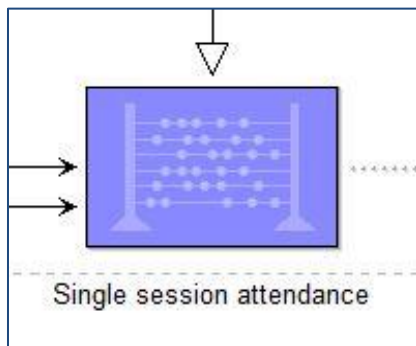


Figure 4. 15 DataSource object with Excel sheet configuration parameters

b) The 3 KPI Objects must be configured as shown below (from bottom up)

KPI 1 (Single session attendance)



Double Click on the KPI object to view attributes with preset entries

Figure 4. 16 Single Session Attendance KPI

This KPI connects to the excel sheet and gets the required fields specified in the “Fields” attribute by clicking the “ Configure Fields” button with aid of a java helper class.

This KPI provides information on the number of participants for each ADOxx Training session.

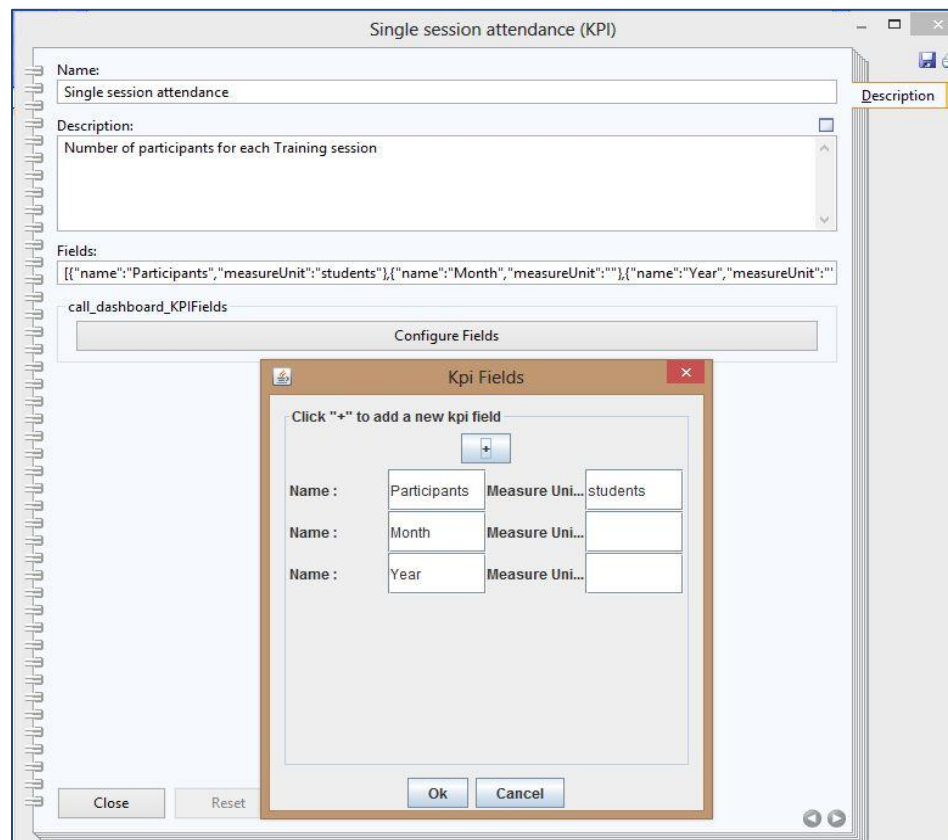


Figure 4. 17 Configuring Single Session Attendance KPI

KPI 2 (Yearly attendance)

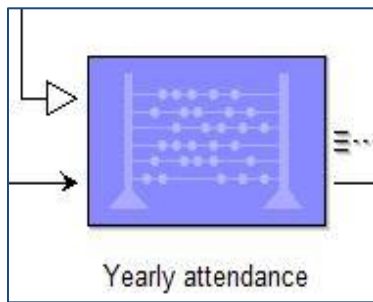


Figure 4. 18 Yearly Attendance KPI

This KPI is generated by the Algorithm “Group by year” and gets the required fields specified in the “Fields” attribute by clicking the “Configure Fields” button with the helper class.

This KPI provides information on the number of participants for a particular year.

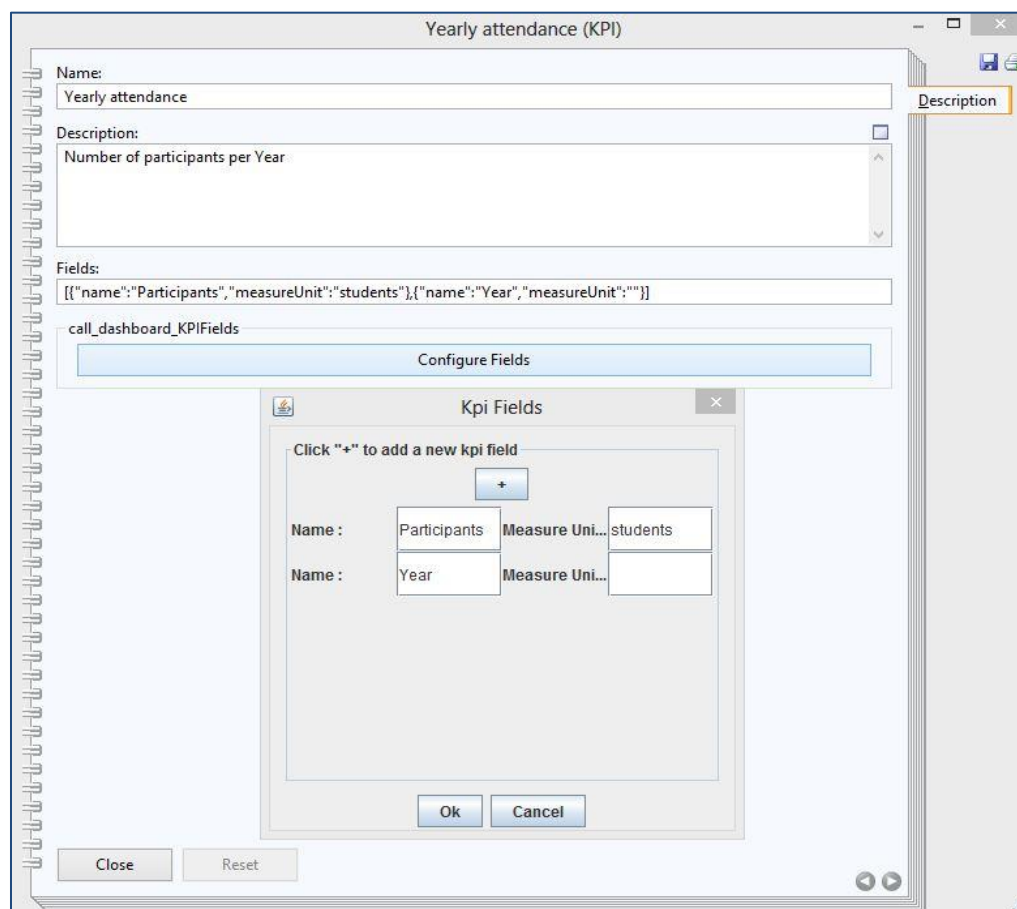
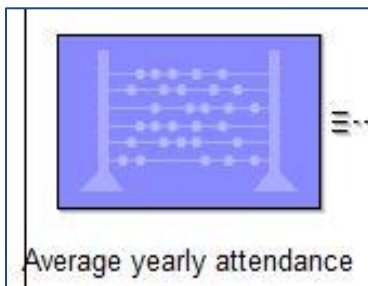


Figure 4. 19 Configuring Yearly Attendance KPI

KPI 3 (Average yearly attendance)

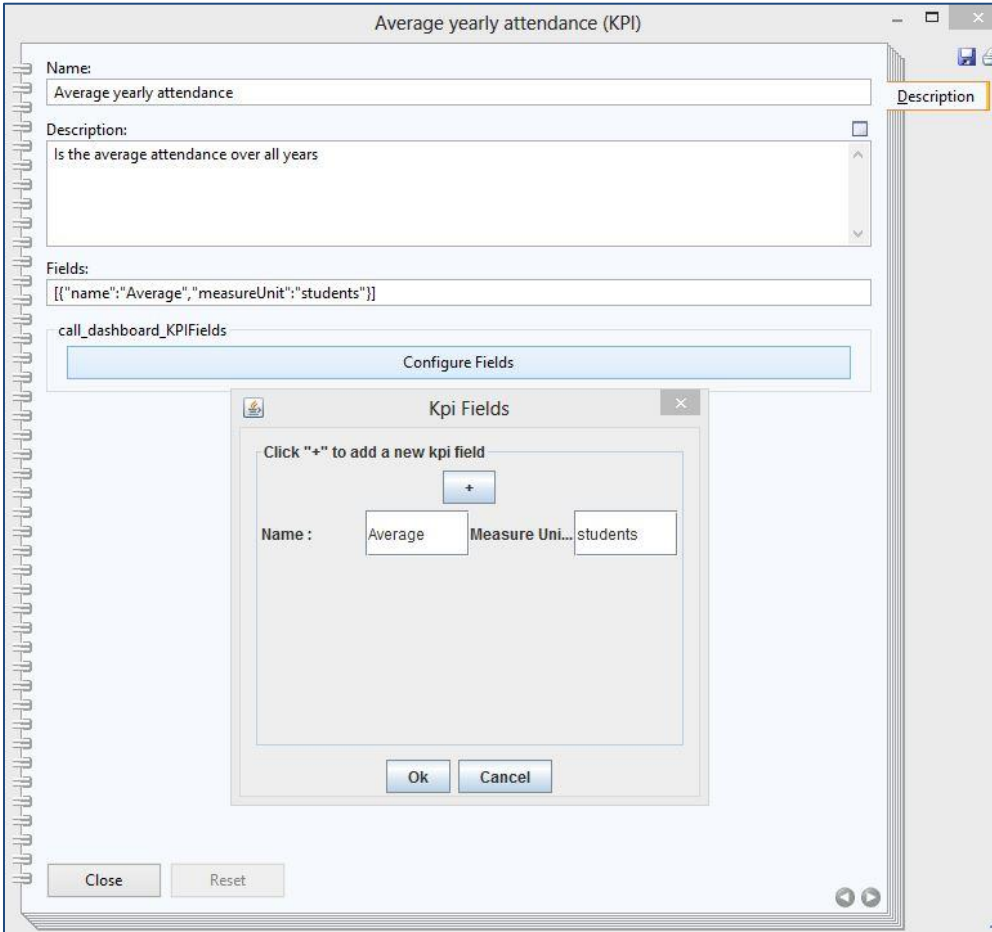


Note: Repeat steps 1), 2), and 3) of “KPI1” above to set up this KPI object with values as shown in the fields below

Figure 4. 20 Average Yearly Attendance KPI

This KPI is generated by the Algorithm “Group by year” and gets the required fields specified in the “Fields” attribute by clicking the “Configure Fields” button with the helper class.

This KPI provides information on the number of participants for a particular year.



Average yearly attendance (KPI)

Name: Average yearly attendance

Description: Is the average attendance over all years

Fields: [{"name": "Average", "measureUnit": "students"}]

call_dashboard_KPIFields

Configure Fields

Kpi Fields

Click "+" to add a new kpi field

+

Name : Average Measure Uni... students

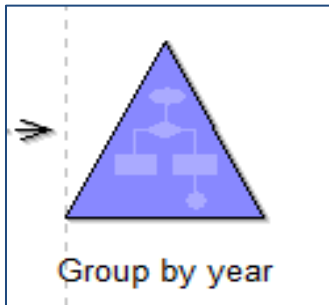
Ok Cancel

Close Reset

Figure 4. 21 Configuring Average Yearly Attendance KPI

c) The 5 Algorithm objects configuration is described as follows:

Algorithm 1 (Group by year)



This algorithm reads the number of participants per single session and groups them according to the years.

The fields consist of the description and a JavaScript code to perform the grouping and provide the result to the connected KPI.

Figure 4. 22 Group by Year Algorithm

Group by year (Algorithm)

Name:
Group by year

Description:
Read the number of participants per single session and group them per year

Code:

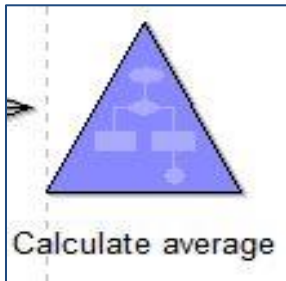
```
var numParticipantSingleSessionValue = requiredKpiValueList[0].value;  
var ret = {  
  columns : ["Participants", "Year"],  
  data : []  
};
```

Description

Close Reset

Figure 4. 23 Configuration of Group by Year Algorithm

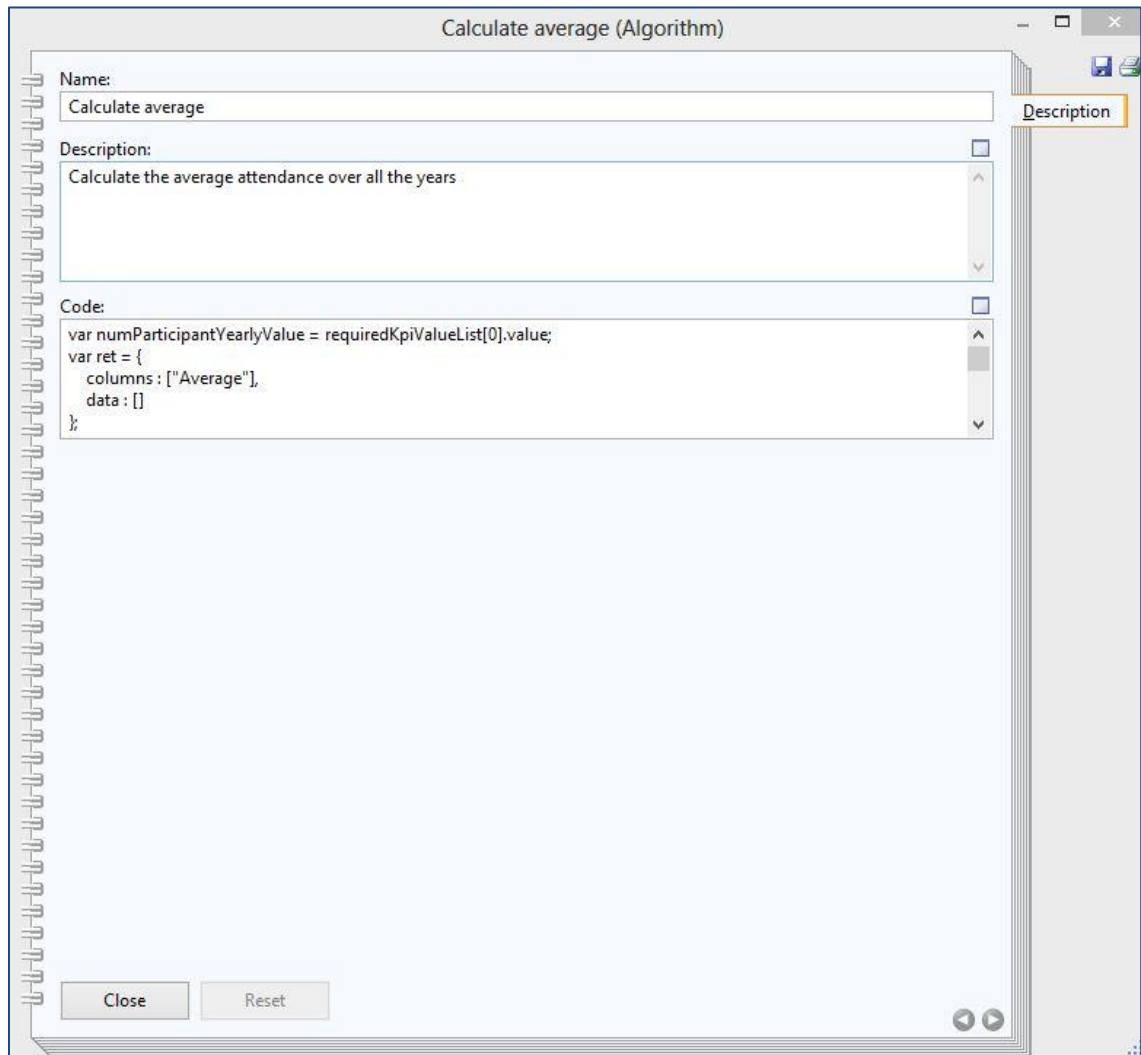
Algorithm 2 (Calculate average)



This algorithm calculates the average attendance over all the years.

The fields consist of the description and a JavaScript code to perform the grouping and provide the result to the connected KPI.

Figure 4. 24 Calculate Average Algorithm



Calculate average (Algorithm)

Name: Calculate average

Description: Calculate the average attendance over all the years

Code:

```
var numParticipantYearlyValue = requiredKpiValueList[0].value;  
var ret = {  
  columns: ["Average"],  
  data: []  
};
```

Close Reset

Figure 4. 25 Configuring Calculate Average Algorithm

Note:

The rest of the 3 Algorithms follows the above set ups and can be viewed by double-clicking the object on the model.

- d) There are 3 Goals to be evaluated in the upper KPI model which required only a goal description to be set usually in a question form.

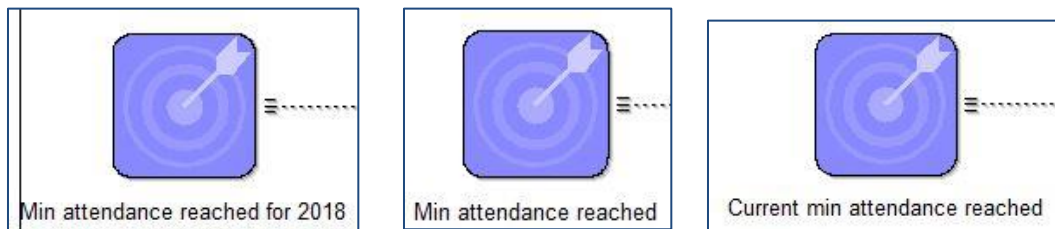


Figure 4.26 Goals to be evaluated

- e) The second part of the KPI model consists of 1 Goal, 2 KPI, 3 Algorithms and 1 Excel Data source as depicted in the diagram below.

In this scenario, the Excel is assumed to be password protected therefore the “**userRequiredInputFieldList**” value is set with a password variable.

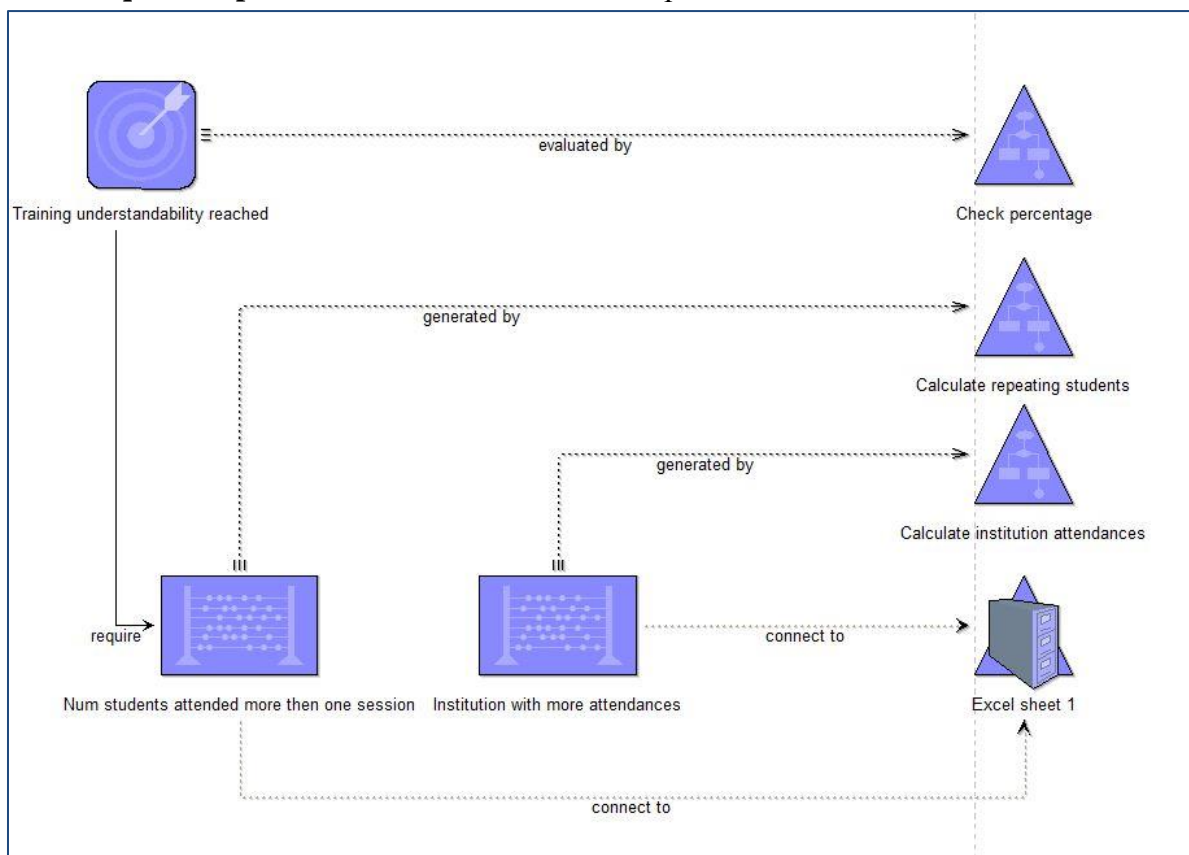


Figure 4.27 KPI Model for Excel sheet 1

f) Now the required JSON file to be provided to the Web Dashboard can be generated and exported. Do this by:

➤ *Click “Extras” on the menu bar → Click “Generate Web Dashboard JSON”*

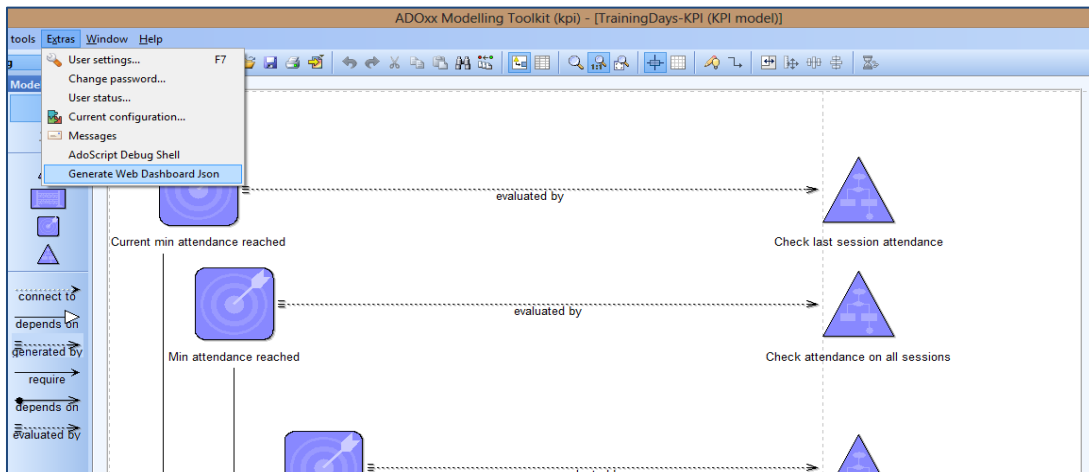


Figure 4. 28 Triggering the AdoScript to Generate the model JSON

➤ *Select the model to be exported → Browse to a location to save exported file*

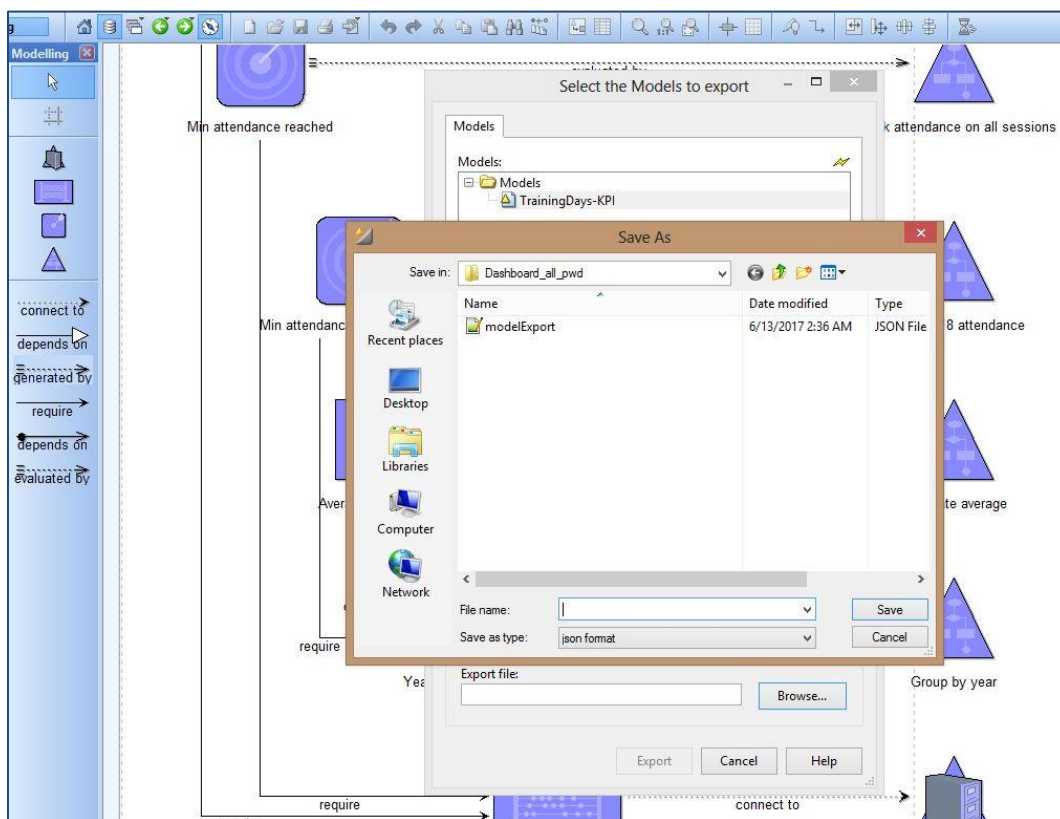


Figure 4. 29 Exporting the generated model JSON

4.2.2. Importing the Dashboard war into Eclipse IDE

1. Import of the Dashboard war file and create new Dashboard project in eclipse.
 - Go to “File” -> Click “Import” -> Locate and select “war file” in the wizard
 - Browse and Select the Web dashboard war file.
 - Give the project a name, select a runtime environment and click “Finish”

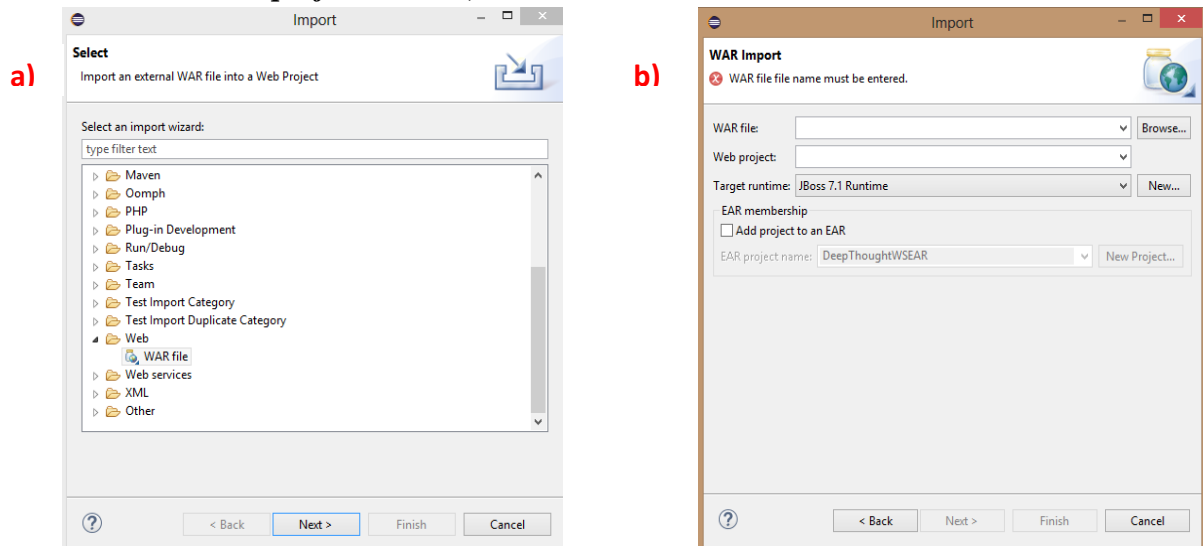


Figure 4. 30 Eclipse IDE import of the Dashboard war file

2. Install an Application Server (if not present) and deploy the project on the server

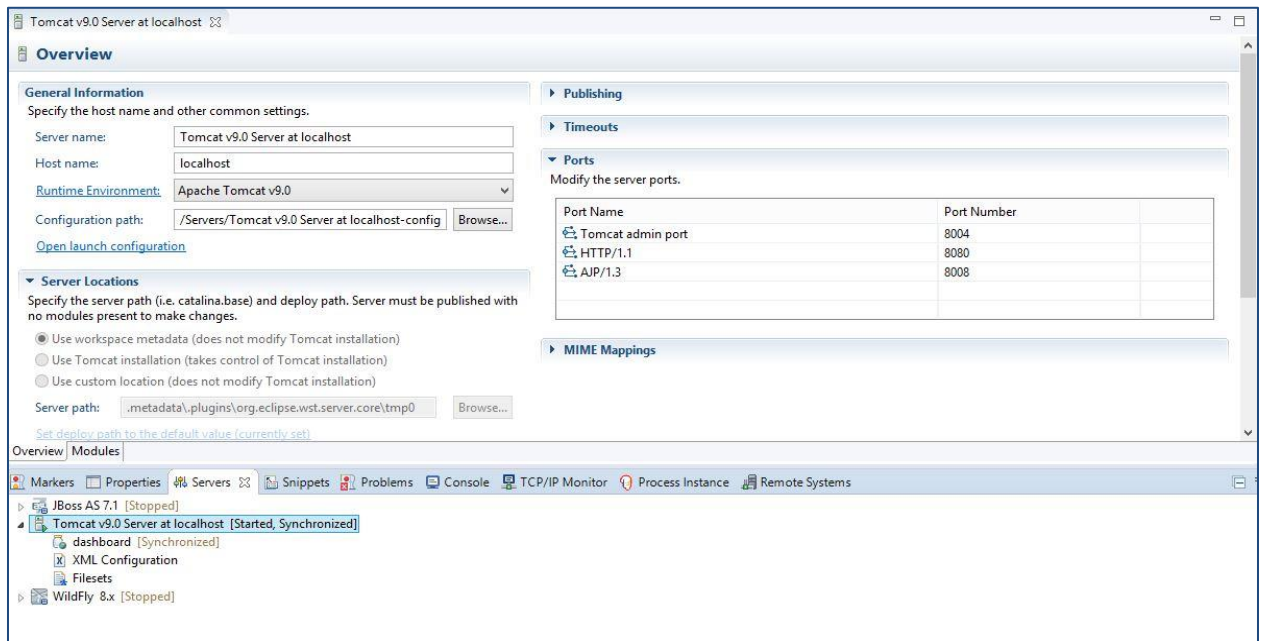


Figure 4. 31 Deploying Dashboard project on Tomcat Application

4.1.3. Configuration and Visualization of the KPI model Result

I. Web Dashboard JSON model import and visualize KPI widgets

1. With the dashboard project deployed on the Tomcat application server as shown above, enter the url : “<http://127.0.0.1:8080/dashboard/dashboard.html#>” in your web browser to view the home page.

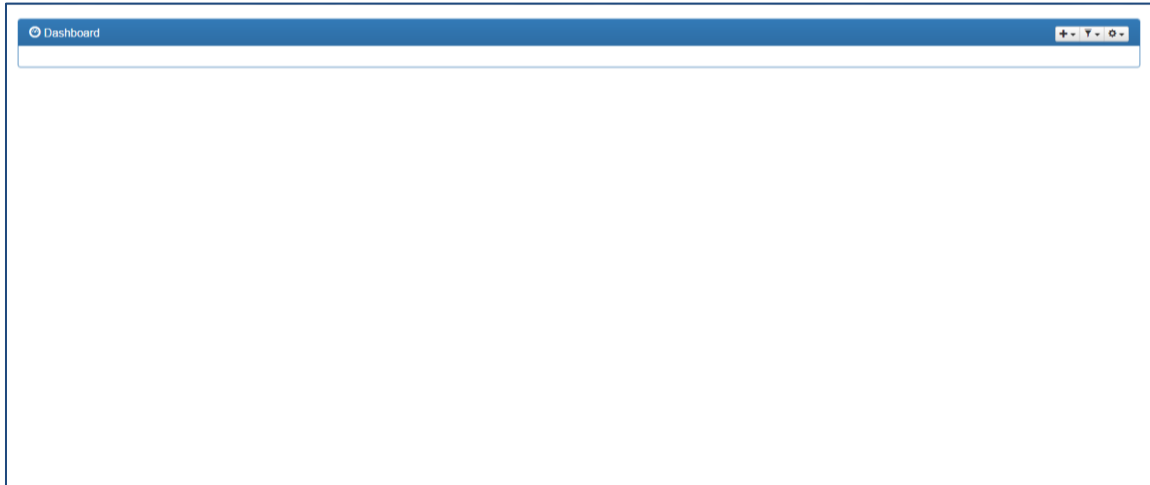


Figure 4. 32 Web Dashboard User interface

2. Click the highlighted button at the top right corner to locate and import KPI model export file

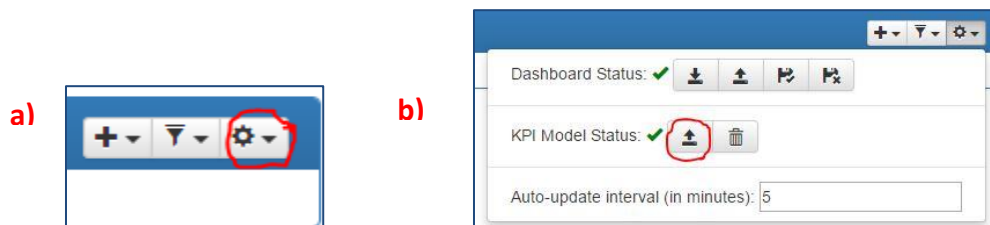


Figure 4. 33 Uploading model Export file on to the Dashboard to visualize KPIs

3. There 3 widgets are provided as shown below. Select any widget to configure and visualize the KPI information as given by the model.

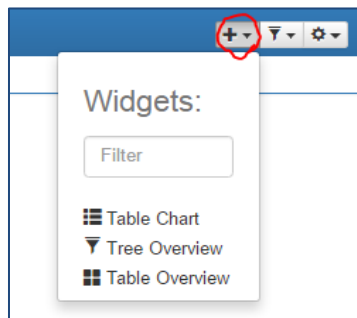


Figure 4. 34 Dashboard widgets

High-level visualization of the dashboard for monitoring and comprehend the data provided below:

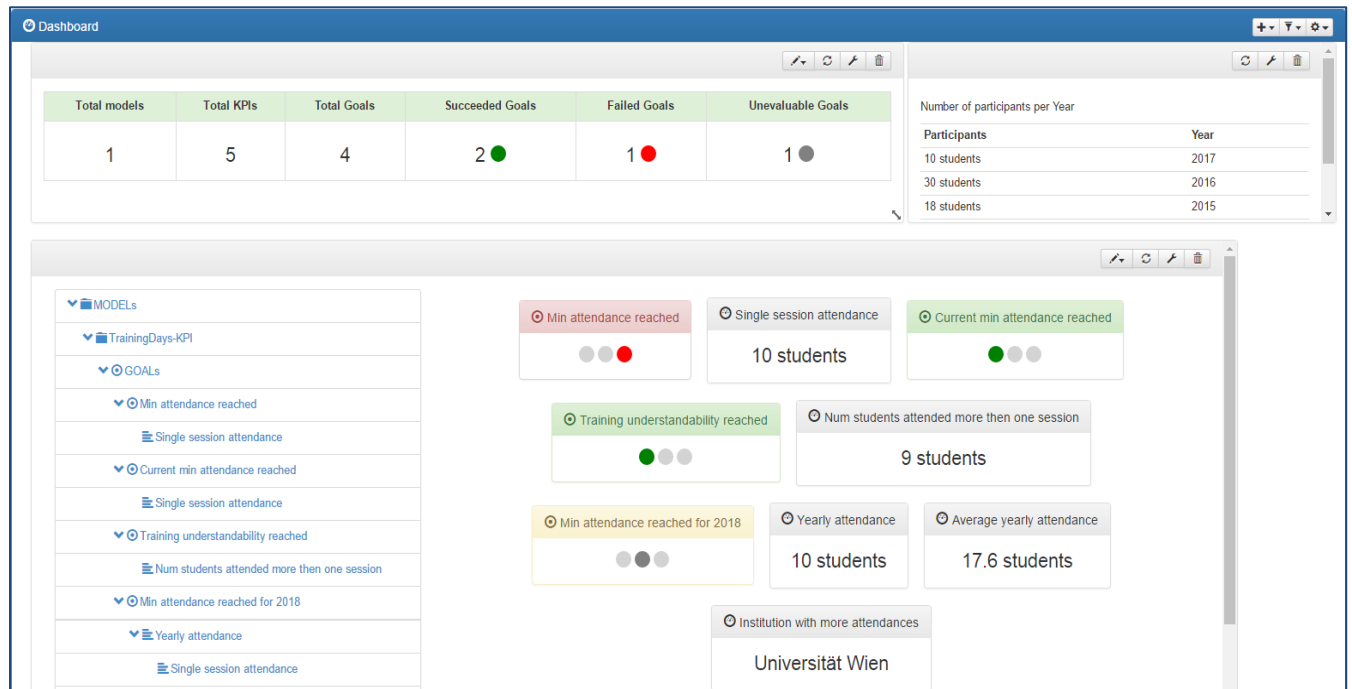


Figure 4. 35 Dashboard Overview of the ADOxx Training Sessions Data

5. Conclusions and Plans for Future Work

This document has provided a detailed description and documentation of the development of the KPI Web Dashboard for monitoring the achievement or otherwise of specific KPIs and goals defined by businesses or organizations. This KPI Dashboard ensures that organizations make well-informed, faster and accurate decisions.

The initial prototype described in this document is limited in scope because it does not explain any detail implementation of how the Dashboard web interface was developed and as well more widgets such as line graph, tachometers, bar charts, etc. can be added to visualize complex data. This can be done by to improve upon this deliverable in order help better understand how the web interface of the Dashboard was developed and as well widen the scope of visualization.

Also, improvement of the DataSource Wrapper by implementation some advanced data sources such as Oracle, NoSQL databases and other cloud-provided databases can be done also with the aim of widening the scope.

References

- [1] ADOxx documentation, available at: <https://www.adoxx.org/live/adoxx-documentation>
- [2] Learn PAd Project Team. D6.2: Learn PAd Simulation Environment: Refined Architecture and Prototype Implementation