OPEN LEARNING ANALYTICS ARCHITECTURE

Annex from the Master-Thesis

An Extensible and Modular Framework for Open Learning Analytics

Note: This document is a printout of the website found at: https://github.com/OpenLearningAnalyticsPlatform/OpenLAP-Architecture/wiki

Prseneted by

Barrios Varela, Oscar Eduardo

341454

First examiner: Prof. Dr. Ulrik Schroeder

Second examiner: Prof. Dr. Horst Lichter

The present work was submitted to Lehr- und Forschungsgebiet Informatik 9 RWTH

Aachen, February 21, 2016

TABLE OF CONTENTS

1. System Overview
2. Documentation Roadmap
3. How a View is Documented
4. Views

    1. Data Collection and Data Storage
        1. Module Views
        2. C&C Views
    2. Analytics Engine
        1. Module Views
        2. C&C Views
    3. Indicator Engine
        1. Module Views
        2. C&C Views
    4. Visualizer
        1. Module Views
        2. C&C Views
    5. Analytics Methods
        1. Module Views
        2. C&C Views
    6. Analytics Modules
        1. Module Views
        2. C&C Views
    7. OpenLAP Web Client
        1. Module Views
        2. C&C Views
5. Rationale
6. Mapping Requirements to Architecture
7. Open Questions

Directory

- Glossary and Acronyms
- Referenced Materials

SYSTEM OVERVIEW

Open Learning Analytics Platform (OpenLAP)

The Open Learning Analytics Platform (OpenLAP) of the RWTH is a project which objective is to provide an ecosystem for Learning Analytics for different stakeholders. The platform will gather data from multiple sources, utilize different methods and apply them to provide insight on the data to the users (Learners, Teachers and Researchers/Developers).

The main users of the project are:

- **Students:** Will utilize the data to provide themselves with insight on course performance, comparison with other students and/or cohorts, receive recommendations on learning materials and subjects, revise achievements and goals in courses and education process. The student will be able to create inquiries and utilize available data from different sources to approach answers to these inquiries.
- **Teachers:** Uses the OpenLAP as a complement of the Learning Management Systems. Utilizes data visualization of the OpenLAP to personalize dashboards with information about student performance, comparison with cohorts, monitoring of activity.
- **Developers/Researchers:** Can create new Analytics Methods and Analytics Goals in the Analytics Modules in order to provide other users new ways of making sense of data and mechanisms to process the data. Can also provide new ways of visualizing the data. Can develop descriptive and predictive models for data analysis as Analytic Methods. Can create modules for the collection of data from third party applications and integrate those modules on the platform. Can use the platform to create custom indicators and provide insight to particular research inquiries.
- **Administrators:** Administrators an perform operations over the administrative and non-functional aspects of the OpenLAP. Maintains the platform, solves issues and allows new Analytics Modules to be used by the users. Has complete access to the functionalities of the OpenLAP.

Functional Requirements

The functional requirements are gathered in base of three main scenarios that derive the main functionalities of the system. The scenarios will be treated as the main Use Cases of the system and will be presented here as such, since they will be the main drivers of the OpenLAP architecture.

SCENARIOS

The scenarios are taken from section 4.2 and section 4.4 of the paper *Toward an open learning analytics ecosystem* (Chatti et. al.). They represent the main usage of the system. They are divided by Teacher, Student and Developer/Researcher Scenarios with the addition of two more: the general use scenarios New Indicator Generation and a system interaction scenario Indicator Data Request.

User Scenarios

User scenarios desrcive those that are executed directly by users of the OpenLAP. They must interact directly with the platform in order to realize this scenarios.

Teacher Scenario

A teacher utilizes an external LMS system to administer courses. The teacher has access to a personalized dashboard of the OpenLAP and from it it can see an overview of courses with certain indicators. The indicators are the main information source to improve teaching. The dashboard provides predefined indicators. Examples of the indicators include: involvement of students on forums, participation of students in course, document usage statistics, progress of students on assignments, learning materials discussed in forums. The teacher can also create new indicators from an indicator editor from which can specify which indicator to use and what visualization to apply to the indicator. The creation of indicators is made available to other users.

Student Scenario

Uses the OpenLAP system to collect data from the university's Learning Management System, MOOCs, Blogs, Social Networks, YouTube, and forums. The student can also chose which activities are collected by the OpenLAP. Data privacy is paramount and by default the student is the only user with access to the collected data. The student can choose what data is public and for how long.

Developer Scenario

Developers and Researchers can develop and register third party application data collection to the OpenLAP. The applications can upload the data to the OpenLAP through a well defined framework. Developers and researchers can as well submit visualization techniques to the platform and these are made available to other users to utilize on their inquiries. Similarly, they can register, through an API, different Analytics Methods to analyze the data and these will be made available to all users. The developer or researcher can also create indicators or use existing ones.

System Scenarios

This scenarios are not directly related to any particular user and can be either performed by any user directly.

New Indicator Generation Scenario

As part of a general user scenario, the New Indicator Generation scenario is available to all registered users of the platform and illustrates one of the most common usages of the OpenLAP. The user selects a goal and enters a question in the question / indicator editor. The OpenLAP system has a component which analyzes the question and suggests similar questions (the question / indicators / metrics component). If the user selects one of the suggested questions, the question / indicator editor will allow the user to select indicators for the question. On the other case if the user creates a new question, the indicators are made available and the user can select which ones to associate to the new question or generate new indicators. When the user selects existing indicators, the OpenLAP makes suggestions of instances of the indicator. The user can select those instances or create a new instance with different method and visualization techniques. Then within the Question / Indicator editor, the user is presented with the visualization code to be rendered. If the user is satisfied, they can use the code to embed it in any application.

Indicator Data Request Scenario

This scenario involves a third party application making a request to obtain visual data from the OpenLAP. The purpose of it is

twofold: to illustrate the interaction of an external system with the OpenLAP and to expose some of the inner workings of the OpenLAP, for this reason, the description of this particular scenario goes beyond only the interactions of the external system with the OpenLAP. In this scenario an indicator request is sent to the OpenLAP from the third party application. The OpenLAP verifies the validity of the request and proceeds as follows:

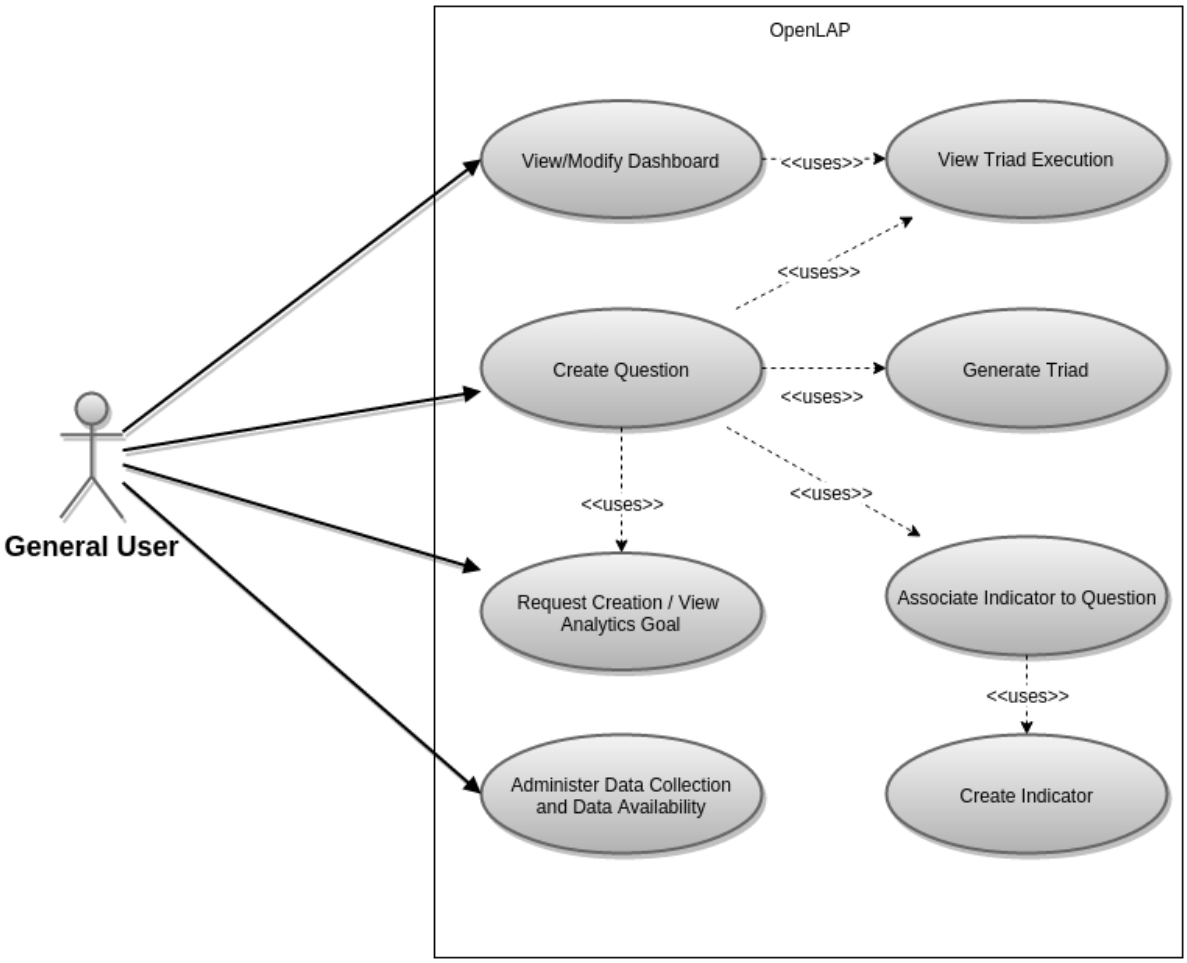
1. The Analytics Engine obtains the indicator reference from the Analytics Modules which contains the Indicator reference, the Analytics Method used, and the particular visualization for the indicator. This grouping of Indicator, Analytics Method and Visualization (along with the mappings of the inputs/outputs) is referred to as a *Triad*.
2. The Analytics Engine obtains the query from the question / indicators / metrics component (Indicator Engine).
3. The Analytics Engine executes the query to the Data Collection and Data Store component.
4. The Analytics Engine uses the Analytics Method to perform data analysis.
5. The Analytics Engine sends transformed data to the Visualizer component.
6. The Visualizer transforms data to a visualization form and returns code for visualization, which is delivered by the Analytics Engine.

USE CASE DIAGRAMS

From the previous scenarios, the following use case diagrams can be derived. They cover most of the main functionalities of the system and are created to provide a complete coverage of the scenarios described above.

General User

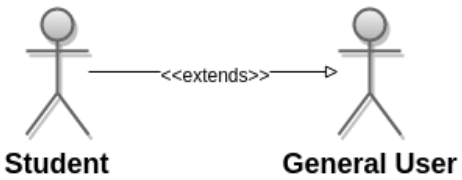
- **View/Modify Dashboard:** The user can see the result of executing the Triads associated with saved indicators on a Dashboard. The user saves the code for Viewing the Triad Execution. Modifying the dashboard implies adding, or removing indicators from it.
- **Associate Indicator to Question:** A user must associate a question to a set of indicators. to answer the question which can be an existing Indicator, a template from an existing Indicator or a new Indicator.
- **Create Question:** Users can create new questions into the OpenLAP system and obtain the view of the execution of the Triad that gets generated by passing the data of an Indicator to an Analytics Method and visualizing with with a particular Technique of the Visualizer.
- **Generate Indicator:** Allows to create a new Indicator from the Rules Engine and the data available.
- **Generate Triad:** Once an Indicator has been Generated, the User can associate the Indicator with an Analytics Method to analyze the obtained data of the Indicator. The Indicator can also be associated to a visualization technique to obtain a visualization of the data. The grouping of the Indicator/Analytics Method/Visualization along with the configuration of the Indicator and Analytics Methods inputs and outputs is called a *Triad*.
- **View Triad Execution:** The triad can be executed once it has been generated. The triad will use the Indicator data on the Analytics Method selected and visualize it with the Visualization technique selected using the mapping configuration saved along with the Triad.
- **Request Creation / View Analytics Goals:** Users can create/Modify/View Analytics Goals (also referred as Analytics Modules). The Administrator allows an Analytics Goal to be used bu other users.
- **Administer Data Collection and Data Availability:** Users can add, modify or remove data collection from third party applications as well as administer who is able to see their collected data and for how long. The Use Case Diagram illustrates the General User Use Cases.



Student

For the main scenarios, the Student will be able to perform the same Use Cases as a general user. The difference with other actors is the particular data that the student has access to, but this is independent of the Use Cases themselves.

The Use Cases Diagram is shown in the figure.



Teacher

Similar to the Student, for the main scenarios, the Teacher will be able to perform the same Use Cases as a general user. The difference with other actors is the particular data that the teacher has access to, but this is independent of the Use Cases themselves. Additionally, the Teacher is able to classify the Analytics Methods into Analytics Goals whenever she or he determines that it is relevant for a particular Analytics goal.

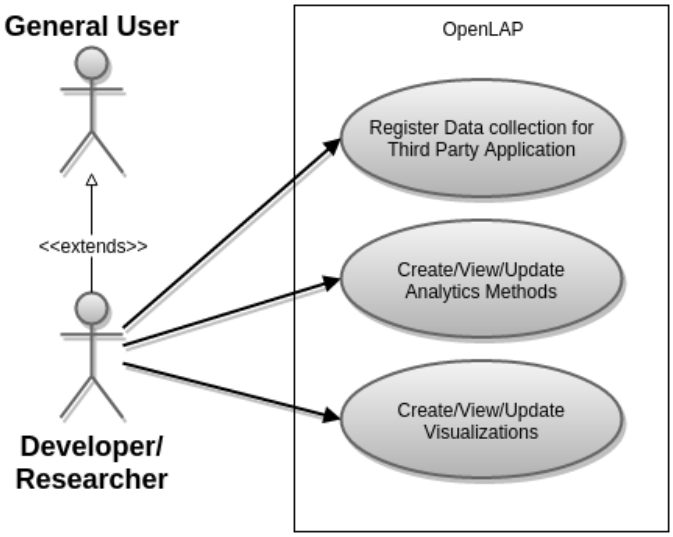
The Use Cases Diagram is shown in the figure.

Developer/Researcher

In addition to the General User Use Cases, the Researcher/Developer can:

- **Register Data collection for Third Party Application:** This Use Case corresponds to the Developer/Researcher Scenario, where a Developer/Researcher submits a mechanism to collect data for a Third Party Application. This is only the registration of a service account and the purpose of registering the external application is to obtain authorization to collect the data.
- **Create/View/Update Analytics Methods:** For the Developer/Researcher Scenario, the Developer/Researcher can create (and modify) new Analytics methods in order to process the data within the OpenLAP.
- **Create/View/Update Visualizations:** Finally, the Developer/Researcher Scenario describes a mechanism for using a customized Visualization for an indicator. The visualization is developed by this user and submitted to the OpenLAP.

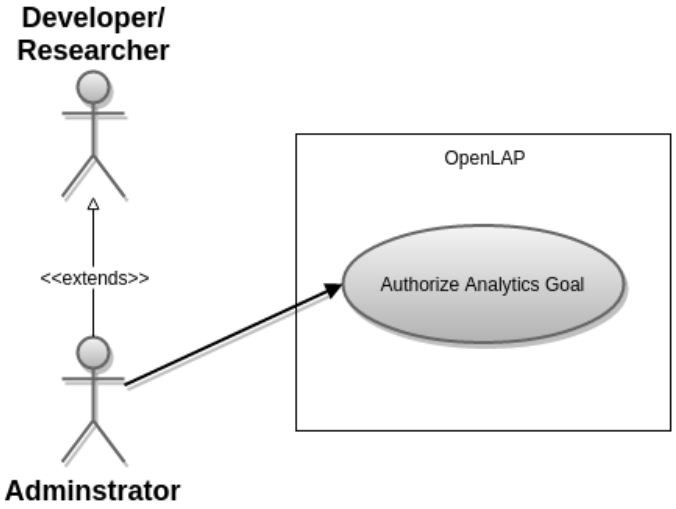
The figure below illustrates this in a Use Case Diagram.



Administrator

Administrators have access to all the functionalities of the OpenLAP. Additionally, they are initially the only users that can authorize Analytics Goals. Analytics Methods can only be related to authorized Analytics Goals.

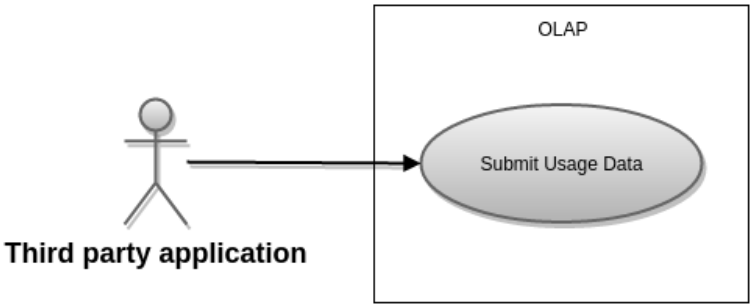
The figure below illustrates this in a Use Case Diagram.



Third Party Application

- **Submit Usage Data:** This is implied on the Student Scenario and the Developer Scenario where a Third Party Application sends data to the OpenLAP.

These are illustrated in the figure as a Use Case Diagram.



USE CASE SPECIFICATIONS

This subsection will present a Use Case Specification Table for each of the Use Cases depicted in the Use Case Diagrams of the previous subsection. This will allows a more comprehensive description of each of the Use Cases and will also assign a number to each one. This information is the basis of the Architecture presented in the rest of this documentation.

The naming convention for the Use Cases is: `OpenLAP_SUC_[Number]`, where `SUC` stands for Scenario Use Case (for the main scenarios, other additional Use Cases developed should use `UC` instead.

OpenLAP_SUC_01: View/Modify Dashboard

Use Case Element	Description
Use Case Number	OpenLAP_SUC_01
Use Case Name	View/Modify Dashboard
Use Case Description	The user can see the result of executing the Triads associated with saved indicators on a Dashboard. The user saves the code for Viewing the Triad Execution. The user can modify the dashboard by adding or removing indicators from it.
Primary Actor	General User
Precondition	The user exists and is logged into the OpenLAP system.
Basic Flow	1. The user enters the OpenLAP Dashboard.

	2. The OpenLAP system executes the Triads and shows the User's Dashboard with all the resulting indicators that the User has added to it.
Alternate Flow: Adding an indicator to the Dashboard	1. The user enters the OpenLAP system dashboard. 2. The OpenLAP System shows the user the Dashboard with the current indicators and the option to add an from a new Question 3. The user selects the option to add an Indicator to the Dashboard. 4. The OpenLAP system continues on OpenLAP_SUC_02: Create Question
Alternate Flows: Remove Indicator from Dashboard	This flow is triggered when the user selects to remove one Indicator from the Dashboard instead of Selection the option to add a new Indicator to the Dashboard. 1. The user selects to remove an indicator from the Dashboard 2. The OpenLAP system asks for confirmation of removal of the Indicator 3. The user confirms removal 4. The OpenLAP system returns to the Dashboard with the indicator removed from it.

OpenLAP_SUC_02: Create Question

Use Case Element	**Description**
Use Case Number	OpenLAP_SUC_02
Use Case Name	General User General Create Question
Use Case Description	The user can create new questions into the OpenLAP system and obtain the view of the execution of the Triad that gets generated by passing the data of an Indicator to an Analytics Method and visualizing with with a particular technique of the Visualizer.
Primary Actor	General User
Precondition	The user exists and is logged into the OpenLAP system. The user is in the Questions / Indicator Editor.
Basic Flow	1. They OpenLAP system presents an interface where the user must select an Analytics Goal and enter a question. 2. The user enters ans Analytics Goal into and a question into the system. 3. The OpenLAP system suggest similar questions related to the entered question and an option to create a new question. 4. The user selects one of the suggested questions. 5. The OpenLAP system shows the indicator instances associated with the question. 6. The user user selects one of the indicator instances. 7. The OpenLAP System continues on OpenLAP_SUC_06: View Triad Execution.
Alternate Flows: The user selects to Create a Question.	This flow triggers at The OpenLAP system suggest similar questions related to the entered question and an option to create a new question. 1. The user chooses to create a new question. 2. The OpenLAP system asks to enter the question and an Analytics Goal 3. The user Selects one of the Analytics Goals for the question. 4. The OpenLAP system continues on OpenLAP_SUC_03: Associate Indicator to Question 5. The OpenLAP system continues on OpenLAP_SUC_05: Generate Triad 6. The OpenLAP System continues on OpenLAP_SUC_06: View Triad Execution with the selected Triad

OpenLAP_SUC_03: Associate Indicator to Question

Use Case Element	Description
Use Case Number	OpenLAP_SUC_03
Use Case Name	Associate Indicator to Question
Use Case Description	The user must associate a question to a set of indicators to answer the question which can be an existing Indicator, a template from an existing Indicator or a new Indicator.

Primary Actor	General User
Precondition	The user exists and is logged into the OpenLAP system. A new Question is being created in the Question/Indicator editor.
Basic Flow	1. The OpenLAP system shows the existing indicators with an option to use them directly or as a template for each. Additionally an option for creating a new Indicator is shown. 2. The user select to use one of the existing indicators directly.
Alternate Flows: Use existing Indicator as Template	This Flow is triggered when the user selects to use an Indicator as a Template in the basic flow.
Alternate Flows: Create Indicator	This alternate flow is show in OpenLAP_SUC_04: Generate Indicator

OpenLAP_SUC_04: Generate Indicator

Use Case Element	**Description**
Use Case Number	OpenLAP_SUC_04
Use Case Name	Generate Indicator
Use Case Description	The user creates a new Indicator from the Rules Engine and the data available..
Primary Actor	General User
Precondition	The user exists and is logged into the OpenLAP system. The user is in the Indicator Generator and has selected to Generate an Indicator.
Basic Flow	1. The OpenLAP system presents the user with the possible rules and the data where the user can apply to the rules. 2. The user picks the indicator rule and assigns the data to apply on the rules. 3. The OpenLAP system generates the indicator and presents the user with possible filters for the indicator to refine the information 4. The user selects which indicators to use over the Generated Indicator (if any).
Alternate Flows	N/A

OpenLAP_SUC_05: Generate Triad

Use Case Element	**Description**
Use Case Number	OpenLAP_SUC_05
Use Case Name	Generate Triad
Use Case Description	Once an Indicator has been Generated, the User can associate the Indicator with an Analytics Module to analyze the obtained data of the Indicator. The Indicator can also be associated to a visualization technique to obtain a visualization of the data. The grouping of the Indicator/Analytics Method/Visualization is called a Triad.
Primary Actor	General User
Precondition	The user exists and is logged into the OpenLAP system. At least an Indicator exist and has been selected. There exists at least an Analytics Method and a Visualization on the OpenLAP system.
Basic Flow	1. The OpenLAP system presents the user with the suggested Analytics Methods as well as Visualizations and asks the user to select an Analytics Method and a Visualization technique and requests for confirmation or change of the mapping. 2. The user selects one analytics method and a Visualization technique. 3. The OpenLAP system presents the user with a default mapping for the data fields of the Indicator to the inputs of the Analytics Method. 4. The user selects the default mapping between the outputs of the Indicator Data and the inputs of the Analytics Method. 5. The OpenLAP system confirms the mapping and validates it. 6. The OpenLAP system presents the user with a default mapping for the data fields of the Analytics Method and the inputs of the Visualization technique and requests for confirmation or change of the mapping. 7. The user selects the default mapping between the outputs of the Analytics Method and the inputs of the visualization technique. 8. The OpenLAP system confirms the mapping and validates it. 9. The user confirms the selection.

Alternate Flow: Mapping Error	This alternate flow is triggered if either the mapping of the fields from the Indicator Data to the Analytics Method or from the Analytics Method to the Visualization technique are not valid.

OpenLAP_SUC_06: View Triad Execution

Use Case Element	Description
Use Case Number	OpenLAP_SUC_06
Use Case Name	View Triad Execution
Use Case Description	The user can submit a Triad for execution once it has been generated. The triad will use the Indicator data on the Analytics Method selected and visualize it with the Visualization technique selected using the mapping configuration saved along with the Triad.
Primary Actor	General User
Precondition	The Triad exists in the OpenLAP system. There is a secure connection between the OpenLAP system and the Third Party Application.
Basic Flow	1. The user sends and indicator data request with a Triad to the corresponding endpoint of the OpenLAP system (The Analytics Engine). 2. The OpenLAP system returns the processed result with visualization to the User application with a status code and a time-stamp.
Alternate Flows	N/A

OpenLAP_SUC_07: Administer Data Collection and Data Availability

Use Case Element	Description
Use Case Number	OpenLAP_SUC_07
Use Case Name	Administer Data Collection
Use Case Description	The User can add, modify or remove Data Collection from external source or application.
Primary Actor	General User
Precondition	The user exists and is logged into the OpenLAP system. The user is in the interface for Administering the Data Collection from external sources or applications. For modifying and removing the Data Collection of an external source or application, the data collection must already be set up for the user. For adding the Data Collection of an external source or application, the module that allows the external source or application Data collection must exist in the system.
Basic Flow: Adding Data Collection from external source or application.	1. The OpenLAP system shows all the current Data Collection from external sources and applications, an option to modify each one, an option to remove each one of them and an option to add a new Data Collection from external source or application. 2. The user selects to add the Data Collection from external source or application. 3. The OpenLAP system shows a list of the available data collection sources for external sources and applications. 4. The user selects one of the sources for data collection. 5. The OpenLAP system presents the user with the authorization screen of the respective source for data collection. 6. The user access the source for data collection and authorizes the OpenLAP system for the collection of data. 7. The OpenLAP confirms the authorization and shows a list of roles to authorize data access to. 8. The user selects the user roles and selects which are authorized to read the data. 9. The OpenLAP confirms the permissions for the data access and the success at adding the external source or application data collection. 10. The OpenLAP finishes the use case.
Alternate Flows: Modifying/Deactivating Data Collection of an external source or application.	This Flow is triggered if the user selects to Modify an existing Data Collection from external source or application. 1. The user selects to modify an existing Data Collection of an external source or application. 2. The OpenLAP system presents the option to deactivate the data collection and a list of the users by role with an option to provide access to read data for each role as well as an option to save the settings. 3. The user makes changes to the roles view persimssions on the Data Collection of an external source or

	application and selects the option for saving the changes. 4. The OpenLAP system confirms the changes done to the Data Collection of an external source or application settings and finishes the use case.
Alternate Flow: Removing Data Collection of an external source or application.	This Flow is triggered if the user selects to Remove an existing Data Collection from external source or application. 1. The user selects to remove an existing Data Collection from external source or application. 2. The OpenLAP system asks for confirmation of the removal of the Data Collection from external source or application. 3. The user confirms the removal of the Data Collection from external source or application. 4. The OpenLAP confirms the removal and finishes the use case.

OpenLAP_SUC_08: Request Creation/View Analytics Goals

Use Case Element	Description
Use Case Number	OpenLAP_SUC_08
Use Case Name	Request Creation/View Analytics Goals
Use Case Description	The User can create/Modify/View Analytics Goals (also referred as Analytics Modules). The Administrator allows an Analytics Goal to be used by other users.
Primary Actor	General Users
Precondition	The User is logged into the system and is on a view to create and update Analytics Goals. For associating Analytics Methods to Analytics Goals, both the Analytics Goal and the Analytics Method must exist. The association can only be done by the Developer/Researcher that updated the Analytics Method or by Administrators
Basic Flow: Creation of a Analytics Goal	1. The System shows the user the option to create a new Analytics Goal. 2. The user selects the option to create a new Analytics Goal. 3. The system asks the user to enter a name and description for the new Analytics Goal. 4. The user enters a name and description to the Analytics Goal and confirms submission. 5. The system notifies the User that the Analytics Goal has been created and finishes the use case.
Alternate Flow: Adding Analytics Method to a Analytics Goal	This scenario can only be done by the Developer/Researcher that updated the Analytics Method or by Administrators. The Analytics Goals can have Analytics Methods attached so they are suggested when generating Triads depending on the Analytics Goal that was chosen there, if any. Otherwise the Triad will show all the possible Analytics Methods with no preferences. 1. The system shows the user the available Analytics Goals and an option to attach Analytics Methods to a selected Analytics Goal. 2. The user selects one of the existing Analytics Goals and selects to attach an Analytics Method to it. 3. The system shows the list of available Analytics Methods that are not yet attached to the Analytics Goal and an option to attach. 4. The user selects one or many of them and confirms the attachment of Analytics Methods to the Analytics Goal. 5. The system confirms the attachment and finishes the use case.

OpenLAP_SUC_09: Register Data collection for Third Party Application

Use Case Element	Description
Use Case Number	OpenLAP_SUC_09
Use Case Name	Register Data collection for Third Party Application
Use Case Description	The Developer/Researcher submits a mechanism to collect data for a Third Party Application. This is only the registration of a service account and the purpose of registering the external application is to obtain authorization to collect the data.
Primary Actor	Developer/Researcher
Precondition	The user exists and is logged into the OpenLAP system. The User is additionally a Developer/Researcher and has permissions to submit Data collection for third party applications. An external service that sends data from an external source or application exists and complies with a data Collection API for the OpenLAP system. The Developer/Researcher is in the interface for registration of Data Collection Applications.
Basic Flow: Register Data Collection Third Party Application	1. The OpenLAP system shows the Developer/Researcher the list of existing registered applications into his account. The system shows an option to register a new Application for Data Collection. Additionally it shows brief instructions for the usage of the registration. 2. The Developer/Researcher selects to register an a new Application for Data Collection.

	3. The OpenLAP system prompts the Developer/Researcher for a Data Collection Application Name and Description. 4. The user enters gives name and description for the Data Collection Application. 5. The system generates a Data Collection Application public/private key to be integrated to it by the Developer/Researcher. The OpenLAP system notifies that the Data Collection Third Party Application is now authorized for sending data. 6. The user takes the Data Collection Application public/private key and finishes the registration process.
Alternate Flows: Modification of Authorization for Data Collection Third Party Application	1. This flow is triggered when the Developer/Researcher selects one of the existing Data Collection Third Party Applications for his account. 2. The OpenLAP system shows the current authorization setting of the Data Collection for the selected Third Party Application. 3. The Developer/Researcher selects to deauthorize/authorize the application. 4. The OpenLAP system asks for confirmation of authorization settings change. 5. The Developer/Researcher confirms authorization settings change. 6. The OpenLAP system show confirmation of Authorization of Data Collection Third Party Application.

OpenLAP_SUC_10: Create/View/Update Analytics Methods

Use Case Element	**Description**
Use Case Number	OpenLAP_SUC_10
Use Case Name	Create/View/Update Analytics Methods
Use Case Description	The Developer/Researcher can implement an Analytics Method using the OpenLAP Analytics Method Framework and submit it to the OpenLAP in order to allow users of the system to use them as Analytics Methods when processing Indicators.
Primary Actor	Developer/Researcher
Precondition	The Developer/Researcher has a private/public key to sign the JAR file to sign the submission of Analytics Method JARs, this public/private key can be generated by the user in the corresponding Alternate Flow for this Use Case. The package to be submitted complies with the Modularity Framework for the OpenLAP system. The Developer/Researcher is logged into the system and is on the Analytics Method submission panel.
Basic Flow: Uploading a Descriptive Analytics Method	1. The OpenLAP system displays a list of the existing Analytics Methods submitted by the current Developer/Researcher, an option for each one to be updated and an option to submit a new method as well as an option to generate a new public/private key pair for submission of Analytics Methods. 2. The Developer/Researcher selects to make a request for submitting a new Analytics Method to the OpenLAP system. 3. The OpenLAP system prompts the user for uploading a signed JAR file, and a Name and description for the Method as well as the type of the Method (Predictive or Descriptive). 4. The Developer/Researcher uploads the signed JAR file to the system. The Developer/Researcher additionally enters the name of the Analytics Method and the Description. The Developer/Researcher also marks the submission as a Descriptive Analytics Method. 5. The OpenLAP system confirms successful upload of the Method with a status code, a validation confirmation, the version read from the JAR file and the name of the method. 6. The user confirms the notification and finishes the use case.
Alternate Flow: Predictive Model Method submission.	This flow is triggered if the user selects to submit a Predictive Analytics Method. 1. The Developer/Researcher uploads the signed JAR file to the system. The Developer/Researcher additionally enters the name of the Analytics Method and the Description. The Developer/Researcher also marks the submission as a Predictive Analytics Method. 2. The OpenLAP system asks for a PMML file for the Predictive model. 3. The Developer/Researcher uploads the PMML file and submits the complete Analytics Method. 4. The OpenLAP system confirms successful upload of the Method with a status code, a validation confirmation, the version read from the JAR file and the name of the method. 5. The OpenLAP system returns to the basic flow: The OpenLAP system confirms successful upload of the Method with a status code, a validation confirmation, the version read from the JAR file and the name of the method.
Alternate Flow: Generation	This flow is triggered when the Developer/Researcher selects to generate a new public/private key pair

of Public/Private key pair for submission of Analytics Methods.	for submission of Analytics Methods. 1. The Developer/Researcher selects to generate a new Public/Private key pair for submission of analytics Methods. 2. The OpenLAP system asks for confirmation of the new Public/Private Key generation. 3. The Developer/Researcher confirms the generation of the Public/Private key generation. 4. The OpenLAP system provides the user with the new Public/Private key generation and confirms that is a current Public/Private key pair for the Developer/Researcher. 5. The Developer/Researcher takes the Public/Private Key and ends the use case.
Alternate Flow: Update Method	This flow is triggered when the Developer/Researcher selects to Update an existing Analytics Method. 1. The Developer/Researcher selects to update one of the listed Analytics Methods. 2. The OpenLAP system continues in the respective procedure of the Basic Flow or the Alternate Flow: Predictive Model Method submission. 3. The OpenLAP system confirms the update of the Analytics Method and finishes.
Alternate Flow: Error submitting method (invalid signature, invalid PMML file, Analytics Method already exists)	This flow is triggered if either the signature used for the submitted files is not recognized by the system or the PMML file submitted for a Predictive Analytics Method is not valid or the Analytics Method already exists. 1. The OpenLAP system shows an error message and returns to the submission panel for the Developer/Researcher to attempt a new submission of the files. 2. The flow continues on the corresponding flow.

OpenLAP_SUC_11: Create/View/Update Visualizations

Use Case Element	**Description**
Use Case Number	OpenLAP_SUC_11
Use Case Name	Create/View/Update Visualizations
Use Case Description	The Developer/Researcher can implement a visualization technique using the OpenLAP Visualization Framework and submit it to the OpenLAP in order to allow users of the system to use them as Visualization technique when processing Indicators.
Primary Actor	Developer/Researcher
Precondition	The Developer/Researcher has a private/public key to sign the JAR file to sign the submission of Visualization frameworks JARS, this public/private key can be generated by the user in the corresponding Alternate Flow for this Use Case. The package to be submitted complies with the Modularity Framework for the OpenLAP system. The Developer/Researcher is logged into the system and is on the Visualization framework submission panel.
Basic Flow	1. The Visualizer system displays a list of all existing Visualization frameworks along with their respective techniques/methods, an option for each one to be updated and an option to submit a new framework as well as an option to generate new public/private key pair for submission of Visualization frameworks. 2. The Developer/Researcher selects to make a request for submitting a new Visualization framework to the Visualizer system. 3. The Visualizer system prompts the user for uploading a signed JAR file, and a Name and description for the framework as well as the list of techniques supported by framework. 4. The Developer/Researcher uploads the signed JAR file to the system. 5. The Visualizer system confirms successful upload of the framework with a status code, a validation confirmation and the name of the framework. 6. The user confirms the notification and finishes the use case.
Alternate Flows:Generation of Public/Private key pair for submission of Visualization frameworks	This flow is triggered when the Developer/Researcher selects to generate a new public/private key pair for submission of Visualization frameworks. 1. The Developer/Researcher selects to generate a new Public/Private key pair for submission of visualization frameworks. 2. The Visualizer system asks for confirmation of the new Public/Private Key generation. 3. The Developer/Researcher confirms the generation of the Public/Private key generation. 4. The Visualizer system provides the user with the new Public/Private key generation and confirms that is a current Public/Private key pair for the Developer/Researcher. 5. The Developer/Researcher takes the Public/Private Key and ends the use case.
Alternate Flows: Updating existing/Uploading new	This flow is triggered when the Developer/Researcher selects to Update an existing Visualization technique of a framework known to the Visualizer system.

visualization techniques for existing Visualization frameworks	1. The Developer/Researcher selects to update one of the listed Visualization techniques of a framework. 2. The Visualizer system asks for a new JAR of the Visualization framework to replace the existing version. 3. The Visualizer system prompts the Developer/Researcher to confirm overwriting the existing framework with the new JAR. 4. The Developer/Researcher confirms and uploads the signed JAR file to the system . 5. The Visualizer system confirms successful upload of the framework with a status code, a validation confirmation and the name of the framework. 6. The Developer/Researcher confirms the notification and finishes the use case.

OpenLAP_SUC_12: Submit Usage Data

Use Case Element	Description
Use Case Number	OpenLAP_SUC_12
Use Case Name	Usage Data
Use Case Description	When a Developer/researches has registered a third party has been registered as a data collector, it can use the OpenLAP in order to submit data to the platform.
Primary Actor	Third Party Application
Precondition	The Data collection for the Third Party Application must be registered and it must be active. Additionally, it must have access to the private/public key pair and must be able to make the submission requests through the provided API. There is a secure connection between the OpenLAP system and the Third Party Application.
Basic Flow	1. The Third Party Application sends a request to the OpenLAP system for submission of a dataset to the appropriate endpoint defined by the Data Collection component of the OpenLAP. The request is signed with the Third Party Application private key. 2. The OpenLAP gives a confirmation of the request stating a successful data submission whit a submission code and a timestamp.
Alternate Flows: Erroneous DataSet sent or invalid signature.	This Scenario triggers if the request of the Third Party Application is not well formed or has incorrect data or if the data is signed with an unknown private key. 1. The OpenLAP system returns an error code with the source of error.

Non-Functional Requirements

- Data Aggregation and Integration: The system should be able to aggregate raw data from heterogeneous sources and different formats.
- Interoperability: Reduce heterogeneity of data to increase interoperability (exchange of information or use information exchanged). It's main usage is for comparison and generalization.
- Specifications and Standards: The system must adopt standards to support interoperability of data and services. Standards include data exchange, models and methods, logging, assessment and privacy.
- Reusability: In order to along itself with the "Open" principles, the system should be able to "reuse, redistribute, revise, remix" components when it is sensible to do so. The use of (open) standards should support this requirement.
- Modularity: The system's architecture should allow for easy integration of components by different developers and allow changes to be made. The platform should have a modular service-oriented approach for easy adaptability.
- Flexibility and Extensibility: The system should be extensible to use new blends of techniques and tools to get information from the available data.
- Performance and Scalability: The system must allow incremental extension of data volume and functionalities. Use big data solutions like MapReduce and NoSQL.
- Usability: The system must integrate GUI design patterns and use appropriate visualizations for better understanding information.
- Privacy: The system must be built always with privacy in mind.
- Transparency: The system must allow users to see that data is not used in unintended ways. Documentation must reflect innards,

state how long is data available and who can access as well of what methods are used on the data.

- Personalization: The system must allow different stakeholders to tailor the OpenLAP to their needs.

DOCUMENTATION ROADMAP

Scope and summary

This documentation describes the architecture of the Open Learning Analytics Platform. The original platform was conceived in the paper *Toward an Open Learning Analytics Ecosystem* by Chatti et. al. 2015. The Open Learning Analytics Platform (OpenLAP) has as main purpose to provide an Open Source hub for providing different Stakeholders the benefits of Learning Analytics. The platform will gather data from numerous external sources and utilize different methods in order to provide insight about the data and improve the learning process for students, teachers, developers, researchers, and administrative staff.

The OpenLAP is designed to be modular, extensible and to procure simple architecture and at the same time utilize standards, design patterns and architectural guidelines in order to support its functional and non-functional requirements (also called "quality requirements").

How the documentation is organized

This architecture document is organized into the following sections:

1. System Overview: This section contains an overview of the system functionalities with a description of the main stakeholders, users, use cases and list of non-functional requirements. The architecture is based in providing blueprints to realize the scenarios described in this section.
2. Documentation Roadmap:It is the present section. This section provides a guideline of the documentation and a reference as well as a summary of the system and useful references.
3. How a View is Documented: This section presents the general layout of the documentation for the different views that each component of the system has. It is based on the example template given on the book *Documenting Software Architectures: Views and Beyond (Bachmann et al. 2011, Addison-Wesley)* which allows ease of reading, simple layout and cohesive presentation.
4. Views: These are the main parts of the architecture. Each one of the main Components of the OpenLAP system are divided in Module, Component & Connector (C&C), and (when necessary) Allocation views. This allows to see each component on a high level of architecture but with sufficient technical depth to provide guidelines for implementation on future iterations over the project.
5. Rationale: This section registers important architectural decisions and the rational behind them. It attempts to convey the constrains and considerations made during the design of the architecture of the system.
6. Mapping Requirements to Architecture: This section provides correspondence between important architectural artifacts and the functional requirements of the system.
7. Open Questions: In this section are listed the tasks that future that can be achieved during future iterations on the OpenLAP.
8. Glossary and Acronyms: A list of important terms, acronyms and definitions.
9. Referenced Materials: Important references used in the design of the architecture.

Places where the documentation is not complete are marked with "[TODO]" or "[TODO:]".

How to update this architecture document

This documentation is meant to be self contained along with the GitHub repository. The documentation here is provided in order to provide other developers and users understanding of the conceptual architecture of the system and forward the development in an understandable manner. The documentation here can (and should) be updated as new components of the system are implemented or added. An initial final version of this documentation will be a section of the Master Thesis *Modularity Framework for Open Learning Analytics Platform* (Barrios, RWTH University, Germany 2015), since it comprises an important part of that work. Further iterations should be aimed to update the wiki version of this architecture.

HOW A VIEW IS DOCUMENTED

Primary Presentation

This section presents the main graphical representations of the view. Whenever a section "key" of the diagram shows the notation used in the diagram. UML is the preferred standard, but any other notation is valid as long as the "key" element comprehensively lists and explain all of it's elements. Other considerations of this section:

- It shows elements and relations of the view and in the vocabulary.
- Can have more than one diagram in order to further explain the view.
- The *architecture cartoon* represents the main piece of the architecture for a view and is supported by the rest of the documentation.

Element Catalog

This section presents and explains each element of the Primary Presentation. It gives further information on the properties of the elements and can extend the notation given.

- Details the elements of the primary presentation with the following sections:
  - Elements and their properties: section for names and properties of the element
  - Relations and their properties: section for relations not shown, otherwise empty
  - Element interfaces
  - Element behavior

Context Diagram

This section displays a context diagram, i.e. the scope of the current view in the larger scheme of the system. The idea is to isolate the view into context of the larger system for better holistic understanding of the system. It also provides a context for the view to compare it with its environment

Variability Guide

This section explains the variability mechanisms for optional components or parts of the view that can change, be configured or be

implemented differently. Also could introduce technological suggestions for known solutions that can support the functionalities depicted through the view. The purpose of this is to show how the pints where the architecture can vary, if needed.

Rationale

This sections conveys the rationale of the significant architectural decisions made in the scope of the view. If any other alternatives where revised, it explains why they were discarded. The section exposes assumptions and constrains as well as influences for choosing the design as it is.

Related Views

Gives references to related and/or children views.

VIEWS

For the organization of the views in this project, a component approach has been taken. Ideally, each core-component is to reside in different nodes and could be easily scaled depending on the needs and usage of the OpenLAP system. Each core-component will have at least a Module and Component and Connector view as well as an Allocation view if deemed necessary (not all of them will provide this view, since the architecture is on a high level enough to not compromise with technologies, and allocation views require a deeper level of understanding of the specifics of the hardware and infrastructure of the available computational resources). Some core-components could show more than one specific view but they will be grouped for ease of access.

The list of core-components of the system are:

1. **Data Collection and Data Storage:** Responsible of obtaining data from external sources and storing it.
2. **Analytics Engine:** Responsible on orchestrating the use of Analytics Methods over the data stored in the Data Storage to provide answers (with their respective visualizations) to the questions established through Analytics Modules from users who utilize the Indicator Engine to access the system.
3. **Indicator Engine:** Allows users to create queries related to learning goals in Analytics Modules through the creation or usage of questions. The users will also be able to select Analytics Methods and Visualizations to use on the data processing and response yielding.
4. **Visualizer:** Provides an extensible selection of implemented visualization options to operate over structured data in order to allows users to obtain different visualizations and graphs of the processed data.
5. **Analytics Methods:** Descriptive and Predictive models to analyze data and a Modularity Framework for extending the available ones by developers. This component will reside with the Analytics Engine.
6. **Analytics Modules:** Grouping of Indicators, Analytics Methods and Visualizations in order provide answer to Learning Goals. This component will reside with the Analytics Engine.
7. **OpenLAP Web Client:** A description of the main components of the Web Client that will allow the users to interact with the OpenLAP system in conjunction. The main interface entry of the OpenLAP.

This selection of the core-component are based on the following guidelines, patterns and recommendations on software architecture:

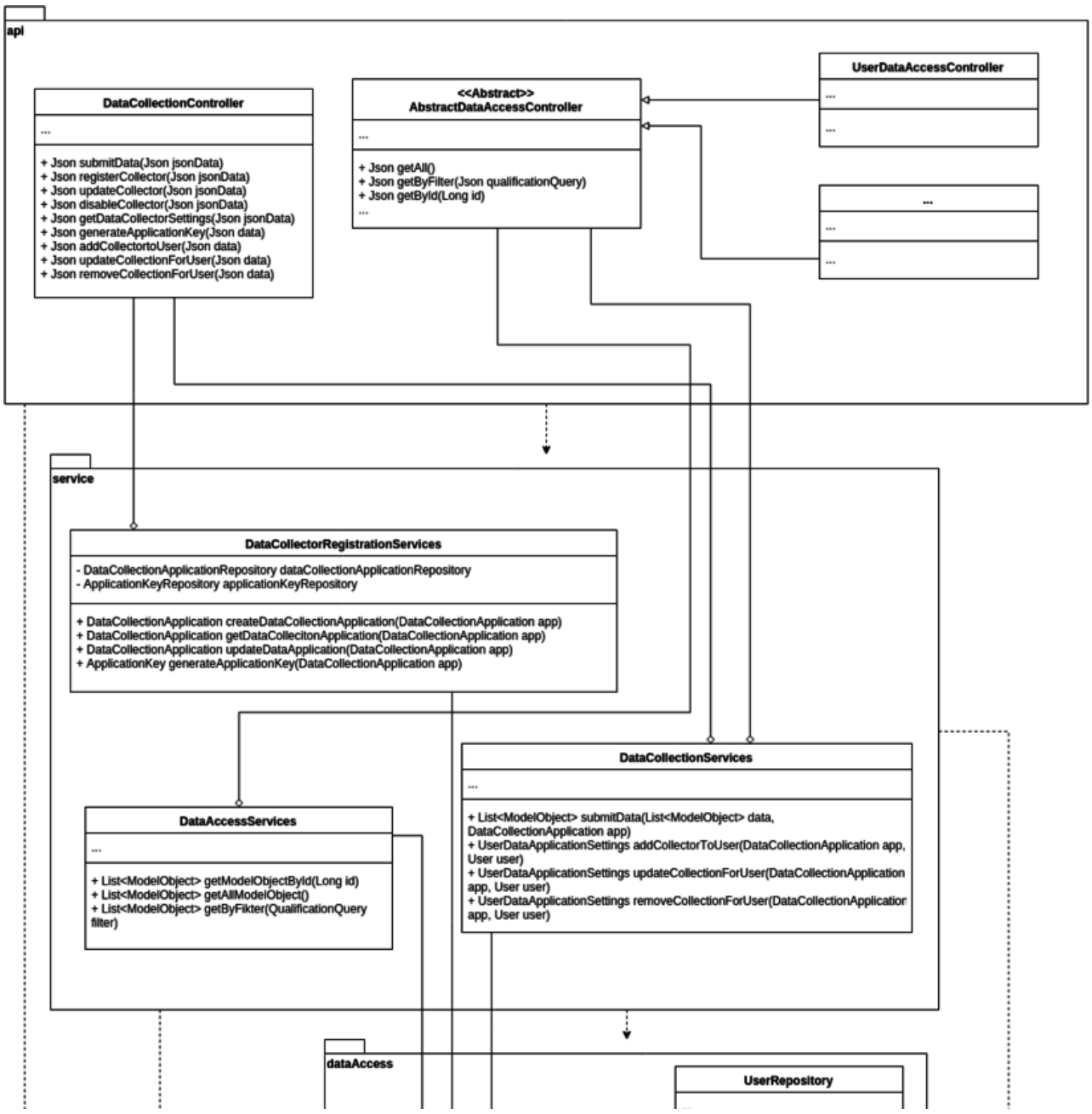
- **High Cohesion, Low Coupling:** By designing the system to use separated units, each core-component must then precisely define interfaces. This does not only ease the interaction with each one of these core-component, but forces the implementation to be scoped to the functionalities of a specific part of the system, therefore, each internal component will focus n provide those functionalities, granting high cohesion. Additionally, each component could function independently, which enables low coupling and permits interchangeability, extensibility and modularity of the system as a whole.
- **"Don't repeat yourself":** A widely adopted principle in system design and architecture, the idea of delegating repeated tasks (or cross cutting concerns) to specialized components (or Aspects in Aspect Oriented Programming) permits to focus the design exercise on the functionalities of the system. Each component assumes tasks like Logging, Security and other non-functional cross cutting concerns to a framework, libraries or modules to be selected by the implementer. Since the non-functional requirements are precisely defined in the System Overview section of this architecture, the tools selected to implement each core-component of the system must provide for solutions that are both compliant with the mentioned non-functional requirements and are also compatible with the other core-components of the OpenLAP. Some guidelines will be provided and where pertinent, detail on the precise tools used to cover this non-functional requirements will be provided.
- **REST:** The Representational State Transfer Architectural style, being more appropriate than pure SOA provides the project on easy to use remote endpoints, independent functionalities, simple programming interfaces (designed to work with HTTP). It also enables decoupling and lightweight application design.
- **Inversion of Control**: By creating the architecture parts to interact with interface instead of concrete classes, it is possible to not only extend the system more easily but also integrate tools for inversion of control and dependency injection. Frameworks like Spring are the mainstream of this design approach and the architecture should be able to exploit their advantages fully and

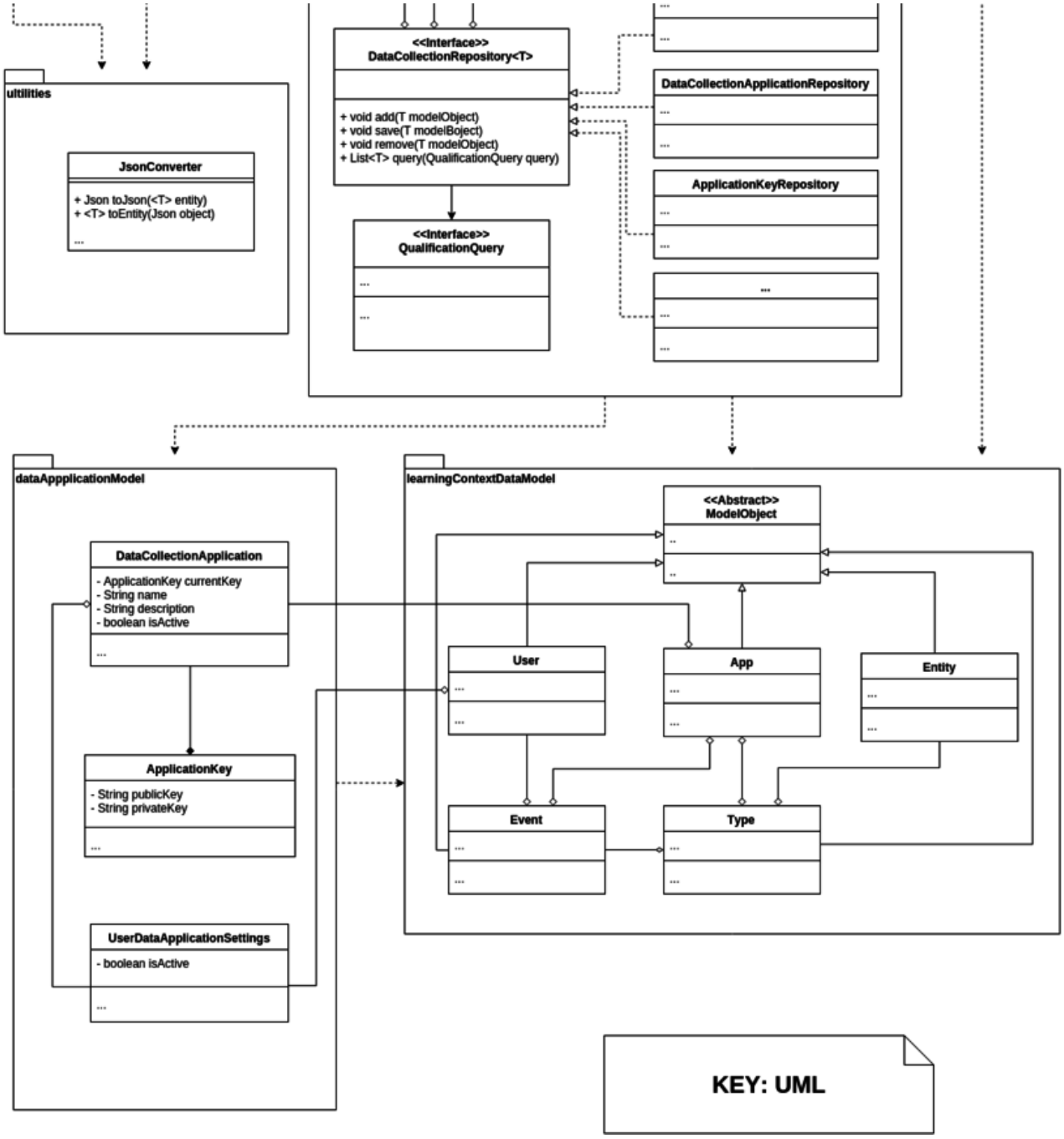
support its modularity and extensibility as well as standardization and best practices non-functional requirements.

- **Standardization and framework usage**: Using a cohesive and standardized design with similar components, architectural approaches across the different macro components allows the different developers to create similar code, increasing its maintainability. Also the adoption of standards through established frameworks and working under the assumption of tested frameworks allows for focusing the design on functional requirements and decoupling them from the non-functional requirements.
- **Extensibility and modularity**: Enabling the developers and researchers to expand the OpenLAP with their own modules, the platform offers APIs that support the extension of the system.

DATA COLLECTION AND DATA STORAGE: MODULE VIEWS

Primary Presentation





Element Catalog

Package: api

This package contains the controllers that expose HTTP RESTful methods. It's purpose is to separate via proxy pattern the access to the rest of the functionality of the macro-component. All the interaction with it is made via JSON and is meant to be implemented exposing it's methods trough HTTP.

- **DataCollectionController**: This class exposes methods for Data Collection Applications that are used to submit data to the

system, it should have a method `void submitData(Json jsonData)` that allows an external registered App to submit data to the OpenLAP via defined endpoint, e.g. `'/api/submitData'`. The methods could vary depending on how the submission process is defined by the developer. This controller also exposes endpoints for General Users to activate and change the settings of the previously registered Data Collection Applications. It also allows to generate the `ApplicationKey` that contains a public/private key pair for a given application if the user is a Developer/Researcher.
- **<<Abstract>> AbstractDataAccessController**: The name is not a reference to the DAO pattern or similar. This is a controller whose purpose is to separate the data access API functionalities.
- **Other controllers**: Other controllers are to expose the querying methods of the specific Learning Context Data Model classes.

Package: service

This is a general service package that exposes the business logic to the controllers. If the controllers are a proxy in the sense of the RESTful approach, this Services is a proxy to the internal logic of the Data Collection, Registration, User Data Collection management and Data Querying of the OpenLAP system.

- **DataCollectorRegistrationServices**: This exposes the services and the business logic for handling the registration of services to the OpenLAP system. It uses objects that are intrinsic to the internal workings and provides exceptions should queries to the database fail. It has two main repositories: the first one to handle the `ApplicationKey` that gets submitted alongside the registration of new Data Collection Applications (and validating that they are valid). The second one a `DataCollectionApplicationRepository` to fetch the available Data Collection applications that exists in the OpenLAP system and their settings.
- **DataCollectionServices**: This method is used to submit data from Data Collection Applications and to handle the user settings regarding what Data Collection Applications they are using and their settings. It should leverage on the `ApplicationKey` public key to grant access to the submission of data to the OpenLAP system by Third Party Data Collection Applications.
- **DataAccessServices**: Ideally, this Service will have as many `DataCollectionRespositories` as there are classes in the `learningContextDataModel` package. The purpose of this class is to realize the querying of relevant data to the main storage of the OpenLAP system and rely it through the classes that extend the `AbstractDataAccessController`.

Package: dataAccess

- **<<Interface>> QualificationQuery**: An interface to realize a filtering mechanism through queries. Implementing classes should take into account the output queries of the Indicator Engine, since those will be the queries that will be received by this macro component and one of its main responsibilities is to be able to process those queries. Implementing classes should be provided in order to be used by the data access repositories.
- **DataCollectionApplicationRepository**: Repository pattern implementation to access the data of `DataCollectionApplication` and `UserDataApplicationSettings`.
- **ApplicationKeyRepository**: Used to access the collection of the `ApplicationKey`s of the system.
- **UserRepository**: Repository pattern implementation to access the `learningContextDataModel` data of `User` type.
- **Other repositories**: Just as `UserRepository` allows for Learning Context Data Model access to data, additional repositories are expected in order to cover the total of the classes on the `learningContextDataModel` package.

Package: utilities

- **JsonConverter**: An utility class to map JSON from/to model layer objects.

Package: dataApplicationModel

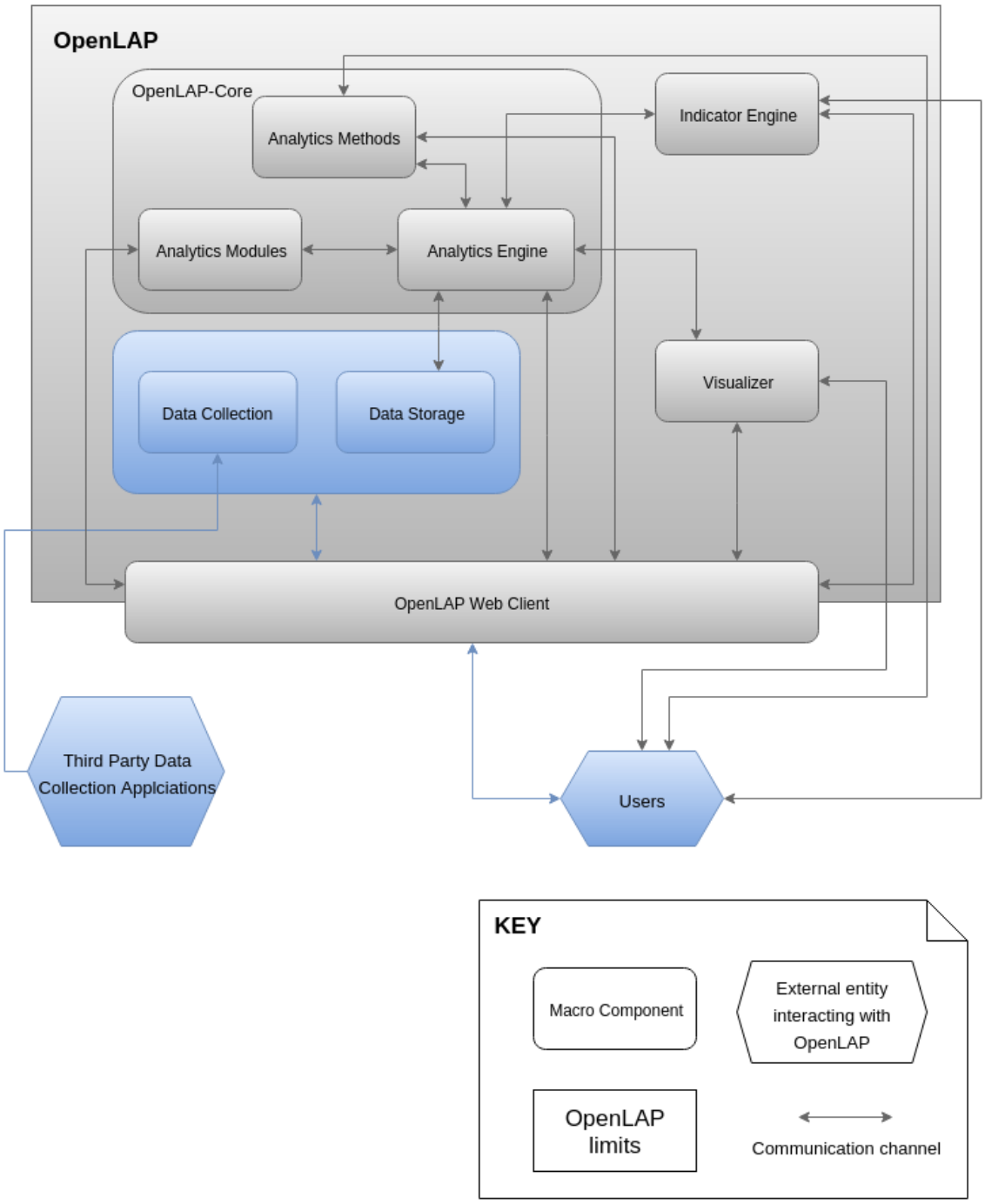
- **DataCollectionApplication**: The metadata of the Third Party Data Collection Application. Sets if is active and contains at least the public key of the application so data submissions are validated for authentication.
- **ApplicationKey**: A generated public/private key pair for the external Applications to use during data submission authentication. Ideally only one key pair is active per Data Collection Application. The system should only store the public part of the key.
- **UserDataApplicationSettings**: Objects of this class are unique per user and represents the list of the Third Party Data Collection Applications that the user is utilizing and their settings.

Package: learningContextDataModel

- **<<Abstract>> ModelObject**: an abstract class that contains metadata of the Learning Context Data Model classes and allows for abstraction and usage on other layers.
- **Learning Context Data Model classes**: Each of this classes represents one of the entities exposed on the Learning Context Data Model. It is the true realization of the concept in the OpenLAP system and should be used as the main approach of data handling over the entire solution. If possible this package should be externalized and integrated to the other macro components for general use.

Context Diagram

As part of the Open Learning Analyics Platform, this macro components is twofold, it allows the interaction of Third Party Data Collection Applications to submit data and allows other parts of the system to interact and access the collected data. A representation within the context of the OpenLAP macro components is given in the figure, highlighting the parts explained in this section.



Variability Guide

- The `utilities` package mos likely can be replaced with an implementation of Gson package.
- The `ApplicationKey` could be separated to provide different objects for public and private keys or it can use a token framework.

Rationale

This macro component uses the general layered architectural style. An API layer exposes the services through controllers that publish endpoints for interactions with users or other macro-components. A Service layer takes care of integrating business logic to the queries and data submission. A Data Access layer groups multiple repositories that allow uncoupled data querying and a Model layer is used to provide containers for all the data objects an Object Mapping to the data fetched from the databases.

The separation between the two model packages is for functional purposes. The `learningContextDataModel` package contains a realization of the Learning Context Data Model described in *Learning Analytics (Lukarov et al. 2014)* and is expected to be heavily used, since it is what allows to realize containers for the bulk of the data for the OpenLAP system. On the other hand, the `dataApplicationModel` package is used for storing relevant information for the user settings regarding registration and usage of Third Party Data Collection Applications.

The Data Collection and Storage of the OpenLAP is a crucial element of realizing the Learning Analytics exercise, therefore, a mechanism for designing a flexible data-store that scales, can cope with new data sources and can provide rapid access to the data consumers and submitter is crucial for the application.

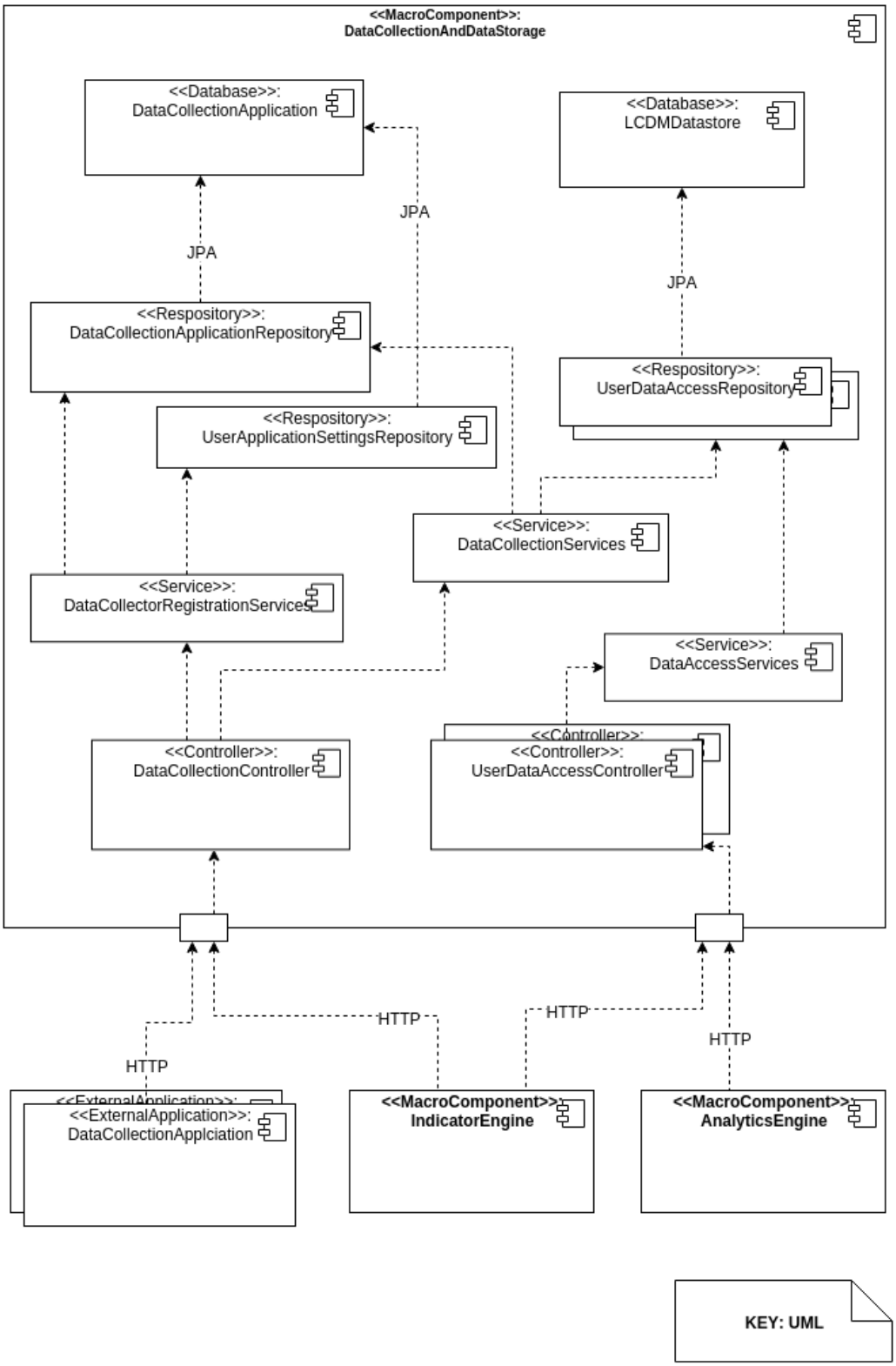
The Data Collection and Data Store component have been conceived as a single element since their responsibilities relate to accessing and providing data for the system.

Related Views

- Section 4.1.2. explains the main Components and Connectors of this macro component and how it is intended to interact with other macro components.

DATA COLLECTION AND DATA STORAGE: COMPONENT AND CONNECTOR VIEWS

Primary Presentation



Element Catalog

Components

- **<<Database>>DataCollectionApplication**: This separate database contains the information of the registered Third Party Data Collection Applications, a relation of users that are utilizing those applications, their status, the public keys and other settings relevant to the data collection application registration. The connection to the database is ideally using JPA but other Database connection mechanisms can be used depending on the implementation and concrete database.
- **<<Database>>LCDMDatastore**: This database contains all the collected raw data of the OpenLAP system. It is the main datastore of the system and will provide for the main functionalities of the Learning Analytics exercise. It is expected to hold a representation in some SQL RDBMS or NoSQL database of the Learning Context Data Model and needs to be performant and highly available. Detains of the precise technologies and implementation of the database itself are left to the implementer. Additionally, this diagram illustrates the database as if resides on the same server as the rest of the components, however that might not be the case on the deployed application for reasons of scalability. This will be restated on the variability guide of this View.
- **<<Repository>>DataCollectionApplicaitonRepository**: This repository provides a data access layer to the properties of the Third Party Data Collection Applications, their settings.
- **<<Repository>>UserApplicationSettingsRepository**: A repository for the settings per user of the different Third Party Data Collection Applications.
- **<<Repository>>Learning Data Model Repositories**: Each repository allows data access to the main Data Store of the OpenLAP system.
- **<<Service>>DataAccessServices**: The Main service of data querying of the OpenLAP system. It provides a logical service proxy point for querying the main OpenLAP LCDM database.
- **<<Service>>DataCollectorRegistrationServices**: This is a service proxy for the creation and modification as well of private/public key pair generation for the Third Party Data Collection Applications and their settings as well as their activation per user and settings.
- **<<Service>>DataCollectionService**: This service provides a proxy for both the data collection functionalities and for managing the User settings of the Data Collection Applications that are being used by the mentioned user.
- **<<Controller>>DataCollectionController**: A REST controller for exposing via HTTP the methods for data collection, data collection user settings as well as activating and deactivating data collection applications per user.
- **<<Controller>>Learning Data Model Controllers**: Each controller will expose HTTP RESTful methods for accessing each of the different Learning Context Data Model entities.
- **<<Component>>IndicatorEngine**: The macro component of the OpenLAP system interacts with both the settings and the data of the Learning Context Data Model. Since the Indicator Engine is the main entry point for users to interact with te system, it must both provide methods to access the settings of the Third Party Data Collection Applications as well as to be able to obtain the queried data that it is intended to provide when generating Indicators.
- **<<Component>>AnalyticsEngine**: The Analytics Engine will query the data whenever Indicator requests are generated in order to pass them to the Analytics Methods and Visualizer, therefore, the Analytics engine must be able to query the central Data Store of the OpenLAP system. It will do this through prepared queries that the controller will rely to the appropriate services and repositories.
- **<<ExternalApplication>>DataCollectionApplciation**: Registered Third Party Data Collection Applications that have valid public keys and are activated are able to perform data submissions to the system through a established HTTP REST API exposed by the DataCollectionConroller.

Connections

- Whenever the connections between components are not specified, it can be assumed that they are remote procedure calls, direct calls or Java Remote method invocation. The concrete implementation will decide the type of specific call.
- Controllers expose their methods through HTTP in a restful manner, through a facade. This allows both other internal OpenLAP components to interact with them as well as allowing any other middleware or clients to connect, given the exposure and authorization. for accessing the methods. It also decouples the functionalities of the macro component and encapsulates them for ease of installation.
- JPA would be the standard data access connection to the Databases.

Variability Guide

- Since connections to the database and the concrete database implementation are left to the developer of this macro component, JPA can change to any sensible technology for guaranteeing the data access. The Data Access Layer is designed so new repositories can be created to interact with the concrete implementations.

Rationale

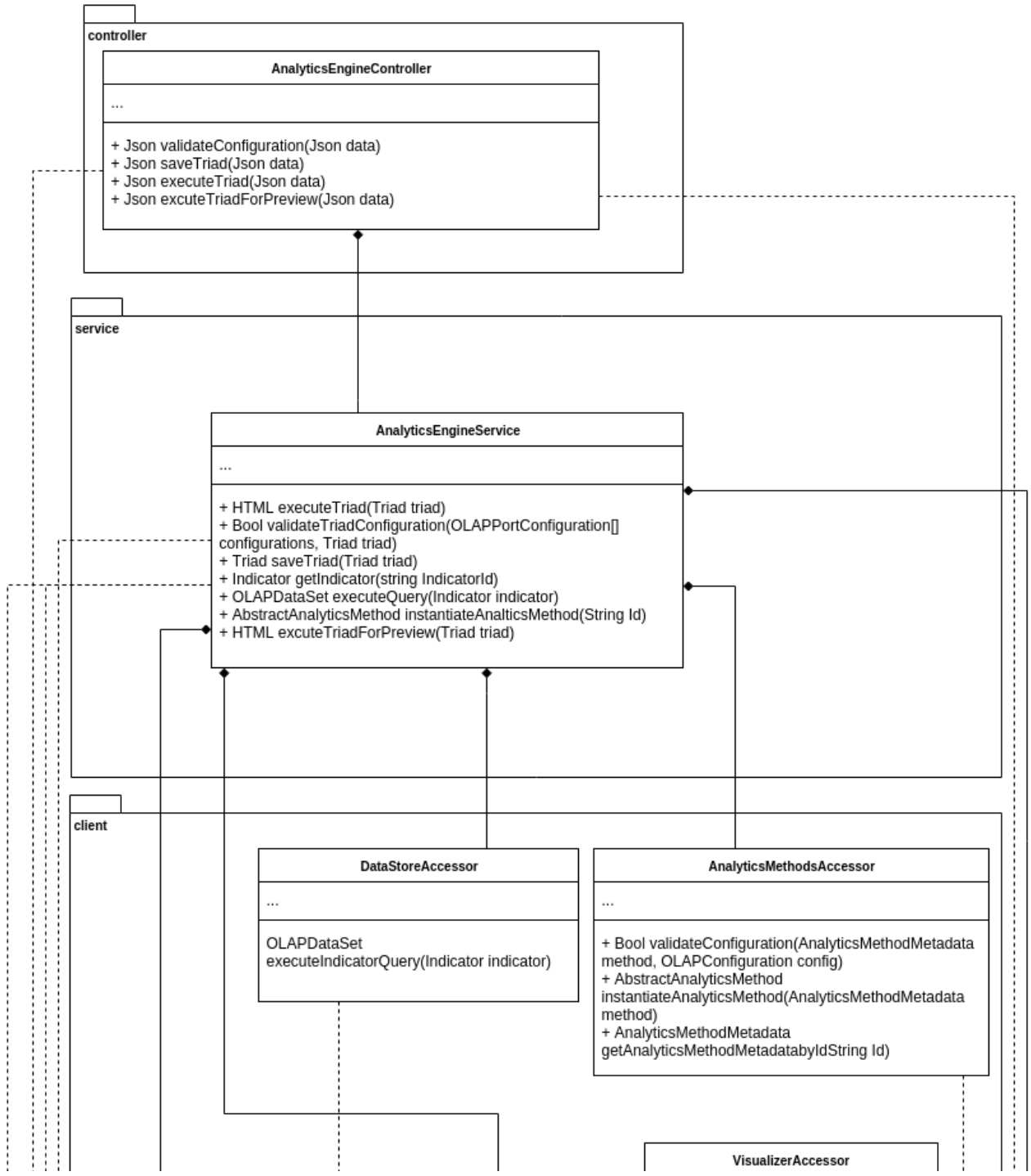
The main idea that this view tries to convey is the RESTful approach to the macro component. Ideally, most of the macro components of the OpenLAP system will expose their functionalities through a set of HTTP endpoints that can be accessed with HTTP requests. The Repository pattern is established so during implementation the concrete database systems can be chosen according to constrains or other requirements. The main functionalities of this macro component are exposed through two controllers: one to handle the Third Party Data Collection Application interaction for submitting data and a second controller to access the data. This separation is based on the users of those functionalities. While Third Party Applications have to perform constant but "spiked" writes in the system, they need a dedicated component to handle those submission requests. At the same time, it needs to be secure and efficient. On the other hand, the other OpenLAP macro components will perform numerous data reads and will need a mechanism for submitting custom queries generated by the Indicator Engine and the Analytics Engine.

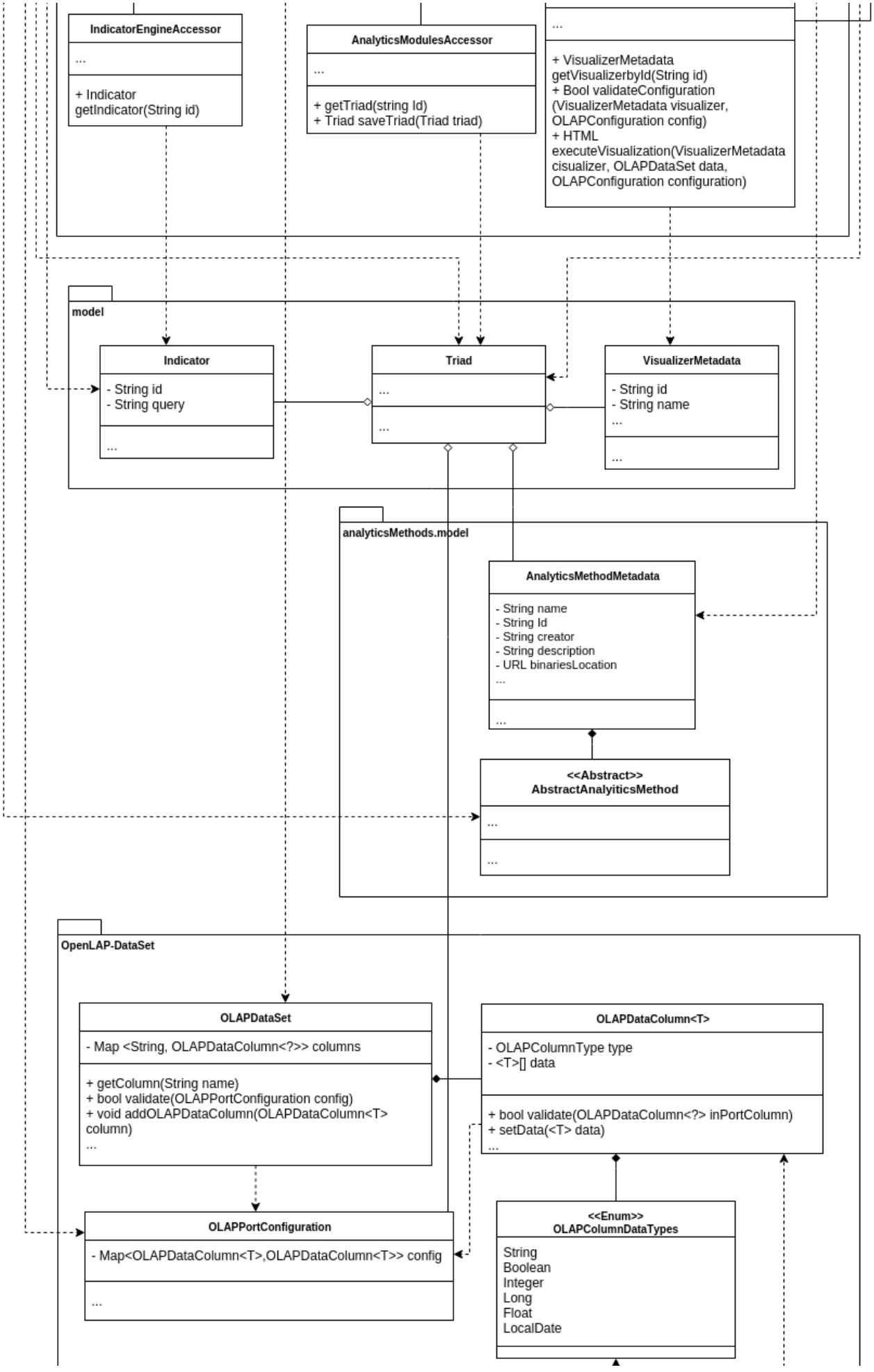
Related Views

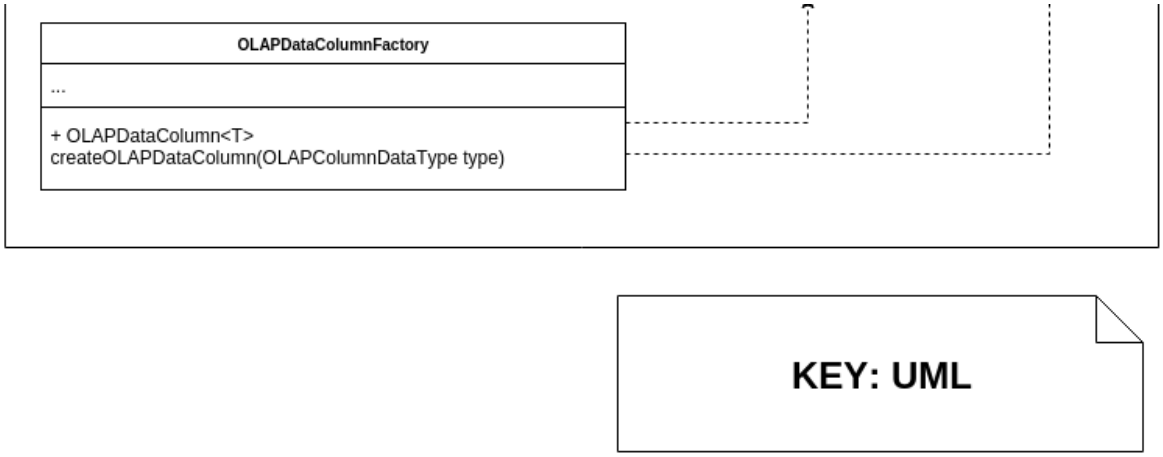
- Section 4.1.1 explains the Module Views of the Data Collection and Data Storage macro component
- Section 4.2 explains the Analytics Engine
- Section 4.3 explains the Indicator Engine

ANALYTICS ENGINE: MODULE VIEWS

Primary Presentation




IndicatorEngineAccessor
...
+ Indicator getIndicator(String id)
AnalyticsModulesAccessor
...
+ getTriad(string Id)
+ Triad saveTriad(Triad triad)
...
+ VisualizerMetadata getVisualizerbyId(String id)
+ Bool validateConfiguration (VisualizerMetadata visualizer, OLAPConfiguration config)
+ HTML executeVisualization(VisualizerMetadata cisualizer, OLAPDataSet data, OLAPConfiguration configuration)
model
Indicator
- String id
- String query
...
Triad
...
...
VisualizerMetadata
- String id
- String name
...
...
analyticsMethods.model
AnalyticsMethodMetadata
- String name
- String Id
- String creator
- String description
- URL binariesLocation
...
...
<<Abstract>>
AbstractAnalyiticsMethod
...
...
OpenLAP-DataSet
OLAPDataSet
- Map <String, OLAPDataColumn<?>> columns
+ getColumn(String name)
+ bool validate(OLAPPortConfiguration config)
+ void addOLAPDataColumn(OLAPDataColumn<T> column)
...
OLAPDataColumn<T>
- OLAPColumnType type
- <T>[] data
+ bool validate(OLAPDataColumn<?> inPortColumn)
+ setData(<T> data)
...
OLAPPortConfiguration
- Map<OLAPDataColumn<T>,OLAPDataColumn<T>> config
...
<<Enum>>
OLAPColumnDataTypes
String
Boolean
Integer
Long
Float
LocalDate



Element Catalog

Package: controller

This package will expose REST HTTP endpoints in order to be used by mainly the `<<MacroComponent>>OpenLAPWebClient` but other clients can connect to the same endpoints. It exposes the main methods of the system as a whole unit.

- **AnalyticsEngineController**: This class will be the primary facade to the OpenLAP system. It holds methods that serialize request and responses to the OpenLAP system. It has a method to validate configurations (mapings between outputs of Indicator or Analytics Methods to inputs of Analytics Methods or Visualization) as well as triad saving methods (a tuple of Idicator / Analytics Method / Visualization) and the execution of the triad itself. It transforms the requests from JSON to/from the specific objects and forwards them to the `AnalyticsEngineService`.

Package: service

The heart of the orchestration component of the Analytics Engine. It holds a class that provides the main workflow of events to execute *triads* and the methods that enable their validation.

- **AnalyticsEngineService**:This is the main orchestration class of the macro component. It has clients to access each of the other Macro Components. When a triad is executed, it gets the Indicator, Analytics Method and Visualization, as well as the OLAPPortConfigurations corresponding to the configurations between Indicator outputs to Analytics Method inputs as well as Analytics Method outputs to Visualization Inputs. It verifies that the mappings of the configuration are correct, executes the query and forwards them to the appropriate Analytics Method and/or Visualization. It also allows to save Triads that are created by the users.

Package: client

These clients or accessors allow the main service to connect to the other macro components of the OpenLAP. The connection can be local in case of the Analytics Methods and Analytics Modules or remote, in case of the other macro components. Each of the

accessors is a proxy of the methods required by the Analytics Engine to the respective controllers on the other macro components.

- **IndicatorEngineAccessor**:Handles the interaction to the API of the Indicator Engine. Principally gets indicators for executing the queries there saved to the Data Store.
- **DataStoreAccessor**: Handles interaction to the API of the Data Collection and Data Store macro component. In particular, runs the queries saved in the indicators and converts the result into `OLAPDataSets`.
- **AnalyticsModulesAccessor**: Handles interaction to the API of the Analytics Modules macro component. The methods in particular allow the Analytics Engine to retrieve and save *triads*.
- **AnalyticsMethodsAccessor**: Handles interaction to the API of the Analytics Methods macro component. It acts as a proxy to the `AnalyticsMethodsService` which resides in the same JVM as this macro component. It allows the `AnalyticsEngineService` to instantiate and run Analytics Methods in the form of `AbstactAnalyticsMethod` objects.
- **VisualizerAccessor**: Handles interaction to the API of the Visualizer macro component. The methods of this client allows to get the specific Visualization, validate `OLAPPortConfiguration`s and execute the Visualization to forward to the requests of triad executions.

Package: model

- **IndicatorReferene**: Object to store the Indicators retrieved from the Indicator Engine macro component. Of particular interest is the query that must be executed against the Data Collection and Data Store macro component.
- **Triad**: This object encapsulates the reference to an Indicator, an Analytics Method (in particular its metadata object) and a Visualizer (also the metadata).
- **VisualizationReference**: Reference to the Visualizer against which the visualization queries will be executed through the respective client.

Package: analyticsMethods.model

This is the same package as the Analytics Methods macro component, and since the Analytics Engine resides in the same running environment, it will leverage on the model to handle the retreival and execution of Analytics Methods metadata and Analytics Method itself (instrances of `AbstactAnalyticsMethod` extending classes). Refer to Section 4.5.1 for a complete description of the model in the context of the Analytics Methods macro component.

- **AnalyticsMethodMetadata**: This is aggregated into the `Triad` and is used to locate the instance of the specific extension of the `AbstactAnalyticsMethod` which corresponds to a specific Analytics Method.
- **<<Abstract>>AbstactAnalyticsMethod**: The extending classes submitted with the Analytics Methods macro component by Researchers and Developers are here used, in particular leveraging on their validation and execution methods in the execution of a triad.

Package OpenLAP-DataSet

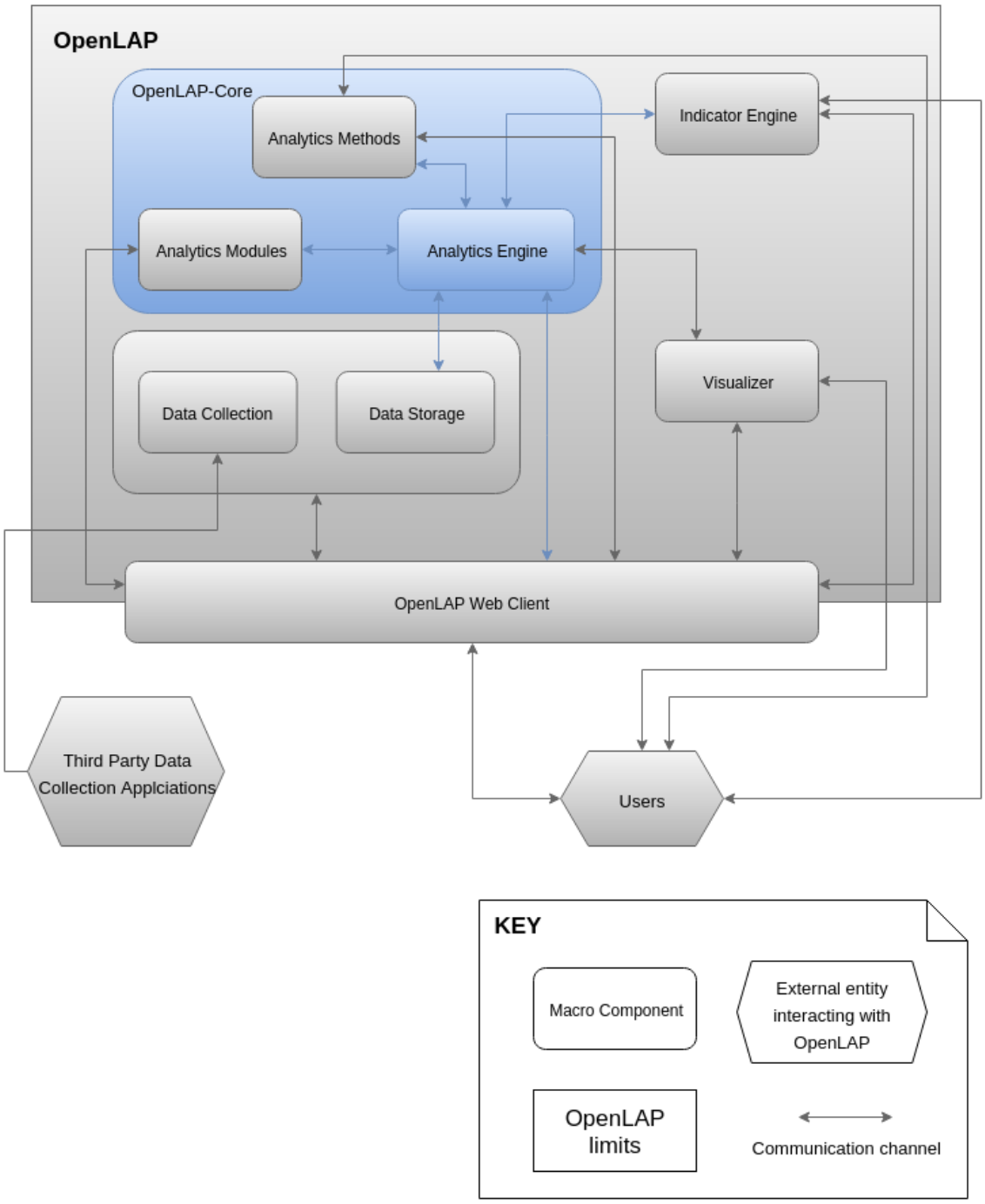
This is a common package for the components that conform the OpenLAP Core (e.g. the Analytics Engine, Analytics Methods and Analytics Modules macro components) as the Analytics Engine. It allows the different macro components in it to have a common data object that is generic and compatible with table-like data.

- **OLAPDataSet**: This is a container for columns. It encapsulates a map between column names and the `OLAPDataColumn`s. When Researchers and Developers are creating new Analytics Methods or Visualizations they must use this data transfer object

in order to allows dynamic type checking from the sources. The possible mappings are (Data from Indicator queries to Analytics Methods or Visualizations

- **OLAPDataColumn<T>**: This class represents a one dimensional column that is to be aggregated by the `OLAPDataSet`. It holds properties for type and array of the data. It also allows to verify that an input is valid for the specific column.
- **OLAPPortConfiguration**: This is a simple mapping between different columns. Its purpose is to support a dynamic checking for the input data from origin to the input port of the using classes. Different configurations can be: Input data form Indicators (Queries) to Analytics Methods or to Visualizations, Analytics Methods to Visualizations, Analytics Methods to other Analytics Methods.
- **OLAPDataColumnFactory**: A factory to create objects of type `OLAPDataColumn` depending on the types available on `<<Enum>>OLAPColumnDataTypes`.
- **<<Enum>>OLAPColumnDataTypes**: An enumerator to be used by the `OLAPDataColumnFactory`. Each entry represents valid column types.

Context Diagram



Variability Guide

- The Analytics Methods to be executed are submitted to the system on the Analytics Method macro component. The `AnalyticsEngineService` instantiates them and executes them.

Rationale

The Analytics Engine is the main orchestration piece and enables the main flow of execution for the triads, which contain references

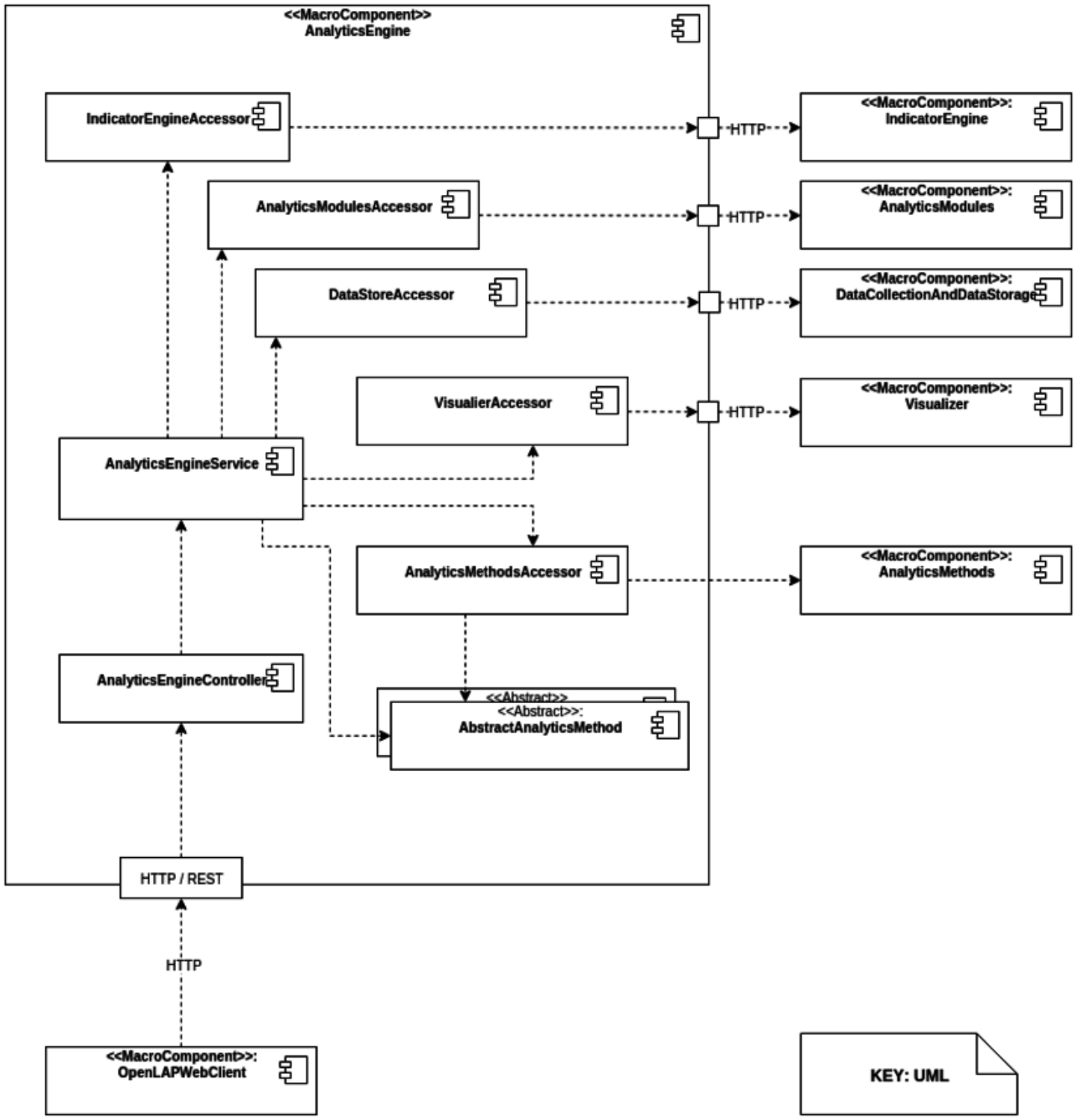
to created Indicators, submitted Analytics Methods and available Visualizations. The run-time of the Indicator and the visualization is external, however the Analytics Methods are run locally in the Same JVM (at least the instantiation). Since the Analytics Engine has to interact with the other components, the modular and extensible nature of the different macro components is then exploited to establish communication through the different REST APIs. At the same time, the main services of the Analytics Engine are encapsulated in the inner controller that enables the Client(s) to execute triads.

Related Views

- Section 4.2.2 exposes the Component and Connector views of this macro component
- Section 4.1 explains the Data Collection and Data Storage
- Section 4.3 explains the Indicator engine
- Section 4.4 explains the Visualizer
- Section 4.5 explains the Analytics Methods
- Section 4.6 explains the Analytics Modules

ANALYTICS ENGINE: COMPONENT AND CONNECTOR VIEWS

Primary Presentation



Element Catalog

Components

- **AnalyticsEngineController**: The controller receives the HTTP REST request from the clients, normally the `OpenLAPWebClient` macro component, but it's RESTful nature can also take request other client requests. The requests are then forwarded to the Service. The purpose of the Controller is to expose the external main methods of the macro component in a proxy fashion.
- **AnalyticsEngineService**: The heart of this macro component. The `AnalyticsEngineService` is responsible for

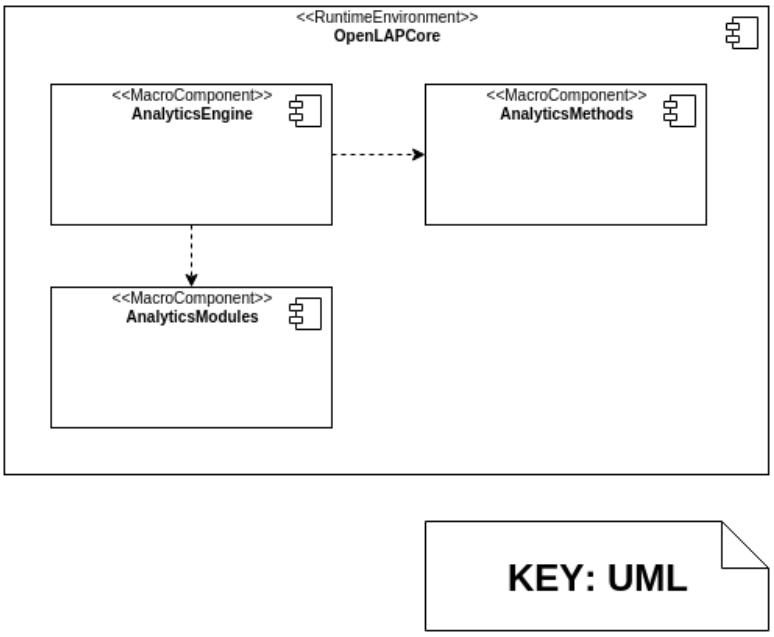
checking the configurations to the triad components, executing and transforming query results and passing them to the Analytics Methods or Visualizations, creating instances of Analytics Methods, executing the Methods and/or passing the result to the visualizations, fetching the result (the HTML generated by the Visualizer) and forwarding it to the controller. The service interacts with multiple clients that allow it to mediate with the other macro components. This component realized the main flow of execution of the Analytics Engine which can be seen in the sub section rationale.

- **IndicatorEngineAccessor**: An internal proxy client to the `IndicatorEngine` macro component via HTTP/REST. It is used mainly for getting access to the Indicators (and specifically the queries to execute).
- **<<MacroComponent>>IndicatorEngine**: The Indicator Engine explained in Section 4.3.
- **AnalyticsModulesAccessor**: An internal proxy client to the `AnalyticsModules` macro component, mainly providing an interface to save *triads*. Note that since the `AnalyticsModules` macro component resides in the same execution environment (JVM) as the `AnalyticsEngine` and the `AnalyticsMethods`.
- **<<MacroComponent>>AnalyticsModules**: The macro component explained in Section 4.6.
- **DataStoreAccessor**: An internal proxy client that helps the service access the `DataCollectionAndDataStore` macro module via HTTP/REST. In specific it allows the service to execute the queries and get the resultset to be passed to either Analytics Modules or directly a Visualization if no Analytics Module is selected.
- **<<MacroComponent>>DataCollectionAndDataStore**: The macro component explained in section 4.1.
- **VisualizerAccessor**: An internal proxy client to the Visualizer via HTTP/REST. It will be used by the `AnalyticsEngineService` to validate configurations and send requests for Visualizations along with the data fetched from the Data Collection and Data Store macro component or that is delivered as output of the Analytics Methods. The resulting HTML code will be then delivered to the client(s) trough the `AnalyticsEngineController`.
- **<<MacroComponent>>Visualizer**: The macro component explained in section 4.1.
- **AnalyticsMethodsAccessor**: An internal proxy client to the Analytics Methods. It enables the `AnalyticsEngineService` to access the available Analytics Methods thus allowing for validation of configurations and execution with the data fetched from the Data Collection and Data Store macro component. Note that this differs somewhat to other access to external macro components The `AnalyticsEngineService` gets direct access to the instances of the Analytics Methods and executes them in the same environment.
- **<<MacroComponent>>AnalyticsMethods**: The macro component explained in section 4.5.
- **<<Abstract>>AbstractAnalyticsMethod**: This are the instances of the Analytics Methods, to which the data obtained from executing the queries that the Indicators wrap will. The `AnalyticsEngineService` uses the instances to validate the configurations and to execute the methods on the data. Then forwards the result to the Visualizer.
- **<<MacroComponent>>OpenLAPWebClient**: The macro component explained in section 4.7. It accesses the methods of the `AnalyticsEngineController` via HTTP/REST and displays is for the users via a web browser.

Connectors

- **HTTP** Calls from `<<MacroComponent>>OpenLAPWebClient`:
- **HTTP** Calls to other `<<MacroComponent>>`: Note that the client components to external macro components i.e. `IndicatorEngine`, `AnalyticsModules`, `DataCollectionAndDataStorage` and `Visualizer` communicate through their own HTTP channel to the corresponding macro components. This means that they establish their own connections so the requests can be handled independently. It is up to the `AnalyticsEngineService` to guarantee the right flow to each one of the components.
- The calls to the `AnalyticsMethods` and the `AnalyticsModules` are don using direct access to those components, since the run environment is shared with the `AnalyticsEngine`.

Context Diagram



Variability Guide

- Whenever the user does not select an Analytics Method, and the configuration is valid, the `AnalyticsEngineService` will pass the collected data of the Indicator query result will be passed directly to the Visualization in the form of a `OLAPDataSet`.
- Some Analytics Methods could potentially communicate with external sources and services to perform their execution, this would mean that they also access outside components in their own protocols. The connections to those external services is allowed and is performed by the Analytics Method instances.

Rationale

The Analytics Engine is the orchestration component and follows the following procedure for the execution of a triad:

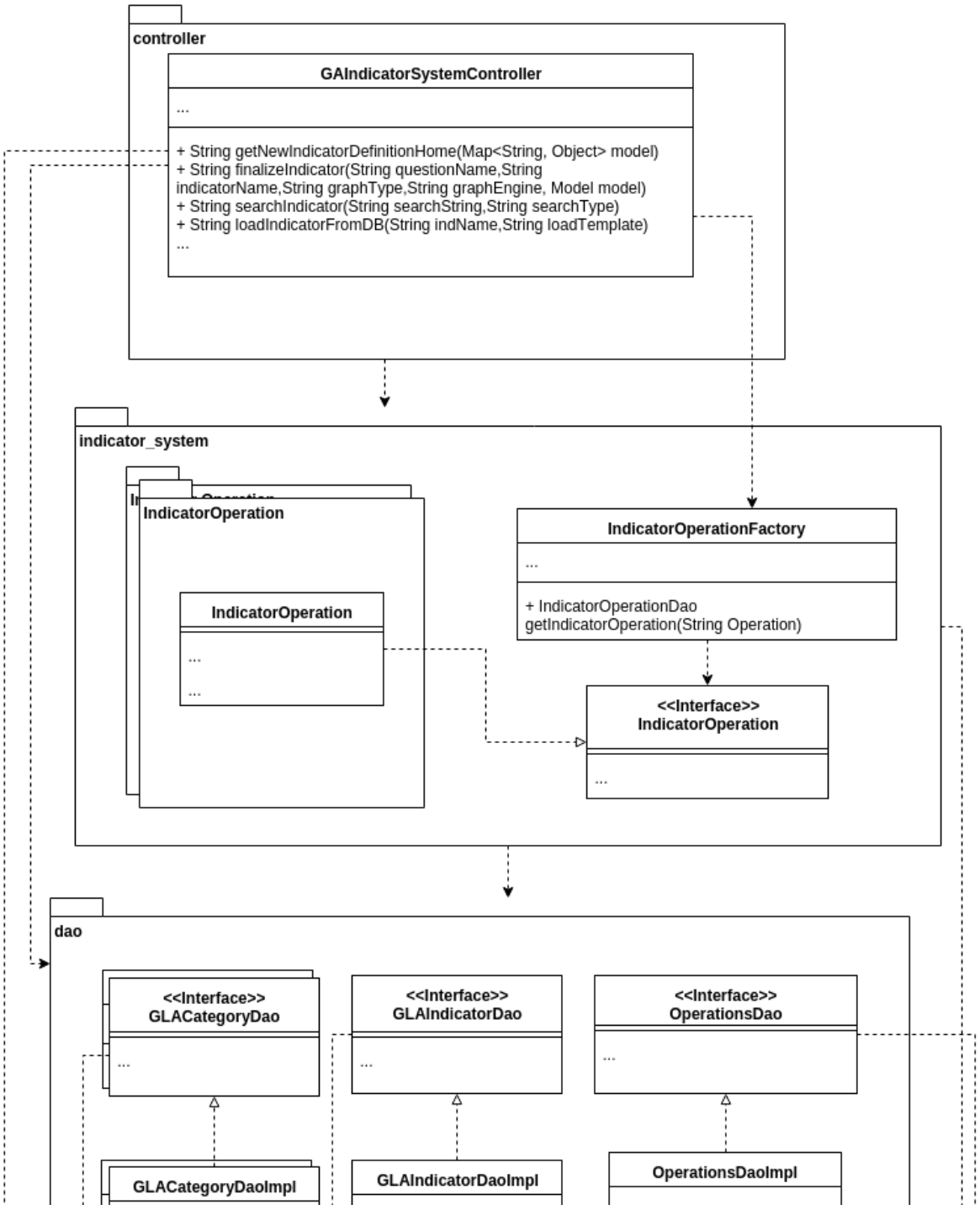
1. The controller receives a request to execute a triad (alongside a configuration), which forwards to the service in the appropriate methods.
2. The service uses the clients to get the the Indicator, Analytics Method metadata and Visualizer Metadata required.
3. The service then checks that the configurations are valid for both the output of Indicator to the Analytics Method input and form the Analytics Method output to the Visualization input. For the Analytics Methods it uses the respective method directly for validation. In the case of the Visualizations, it access them through HTTP using the respective proxy.
4. If the configuration is valid, then executes the query of the Indiciator against the Data Collection and Data Storage macro component.
5. The result is converted to `OLAPDataSet` object and forwarded to the Analytics Method instrance. This is done locally, since the run time is the same.
6. The result of the analysis of the Analytics Method (also a `OLAPDataSet` object) is sent as a request to the Visualization for processing.
7. The Visualization returns the HTML code which is then returned to the controller. Additionally, the Analytics Engine macro component should be able to filter based on both the Analytics Goal and fetch the input methods for suggestions when the user selects the Analytics Method and the Visualizations.

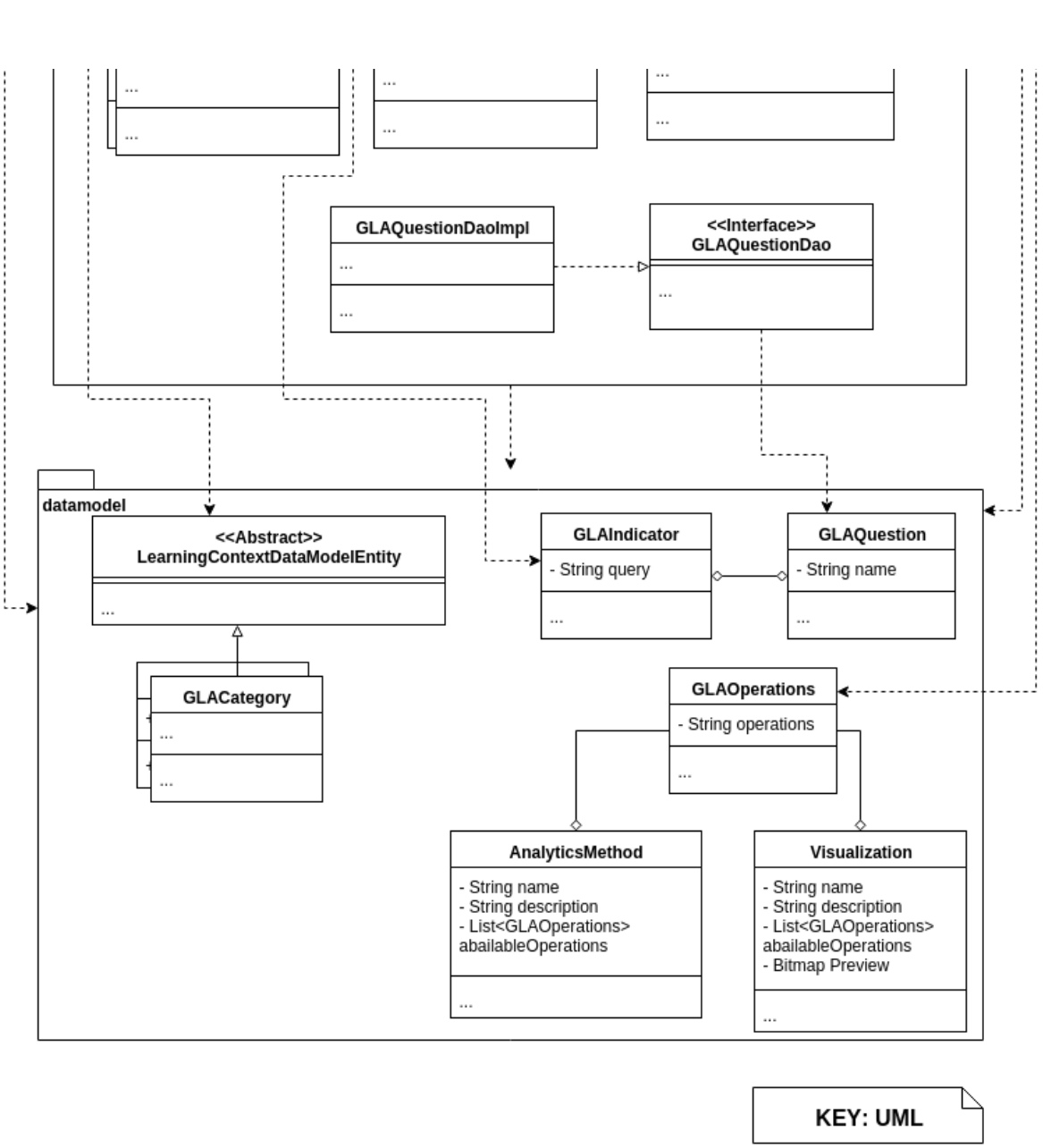
Related Views

- Section 4.2.1 exposes the Module views of this macro component
- Section 4.1 explains the Data Collection and Data Storage
- Section 4.3 explains the Indicator engine
- Section 4.4 explains the Visualizer
- Section 4.5 explains the Analytics Methods
- Section 4.6 explains the Analytics Modules

INDICATOR ENGINE: MODULE VIEWS

Primary Presentation





Element Catalog

Package: controller

This package is in charge of holding the main RESTful API endpoints of the entire system system. Classes to handle endpoints for support operations such as validation, user authentication and statistics should be integrated in this package. The OpenLAP Web Client of the complete application should communicate mainly with this module's controllers

- **GAIndicatorSystemController**: This controller will expose the main internal and external methods of the OpenLAP system to be used both through HTTP requests and to/from the OpenLAP Web Client that will serve as the main user interaction place to the platform. It exposes all the pertinent Indicator Engine options to Create, Update and View Indicators for he users. It also allows to perform CRUD operations on the Questions as well as linking indicators to them. The indicator engine also provides functionalities to create filters over indicators and generate the proper code based on the indicator request code it returns to the

user. The Controller must have accessor clients for both the Analytics Methods and the Visualizations in order to expose the available analytics methods and visualizations when instantiating the Indicator (associating the Indicator to an Analytics Method and a Visualization). It will access the usable Analytics Methods and Visualizations through a JSON object that contains the operation of the indicator, in which case the clients will provide the possible Analytics Methods / Visualizations.

Package: indicator_system

This package holds the "business logic" of the Indicators and Questions as well as providing the drools rules and the interpreter classes for each set of rules that the system applies for eventual generation of the SQL queries that will get forwarded to the Data Collection and Data Storage macro component. It is also the implementation point for new rules to be added on the system and the preprocessing of stored indicators for the controllers to use. This package holds support classes for providing these services to the controller and sub packages for each Operation that is to be defined for the indicators.

- **<<Interface>>IndicatorOperation**:Interface for the Abstracting the Indicator Operations. It will be used by the Factory given what the controller requires in order to handle the appropriate instance of Indicator.
- **IndicatorOperationFactory**: This factory handles the passing of the correct instance of IndicatorOperation to the controller and trigger the rules that are relevant.

Package: indicatory_system.IndicatorOperation

This package is meant to be replicated for each Indicator Operation that is to be made available. Whenever new rules are added, a corresponding IndicatorOperation package with its own Dao and implementation should be created. This also implies that the main controller in the package `controller.GAIndicatorSystemController` should be modified in order to accommodate the new rule-set.

- **IndicatorOperation**: Implementation of the aforementioned IndicatorOperation. Each package that defines new rules must provide an implementation to maintain the design.The responsibility of this class is to trigger the rules based on what is passed down by the `GAIndicatorSystemController`. The rules must be provided in this package as well.

Package: dao

This package holds DAO and Implementation of both Indicator, Question and Operations data and Learning Context Data Model related classes. At least an implementation of each of the DAOs corresponding to entities of the Learning Context Data Model should be able to use the package `learningcontextdata` and use the `DataStoreAccessor` to load the model from the macro component Data Collection and Data Storage.

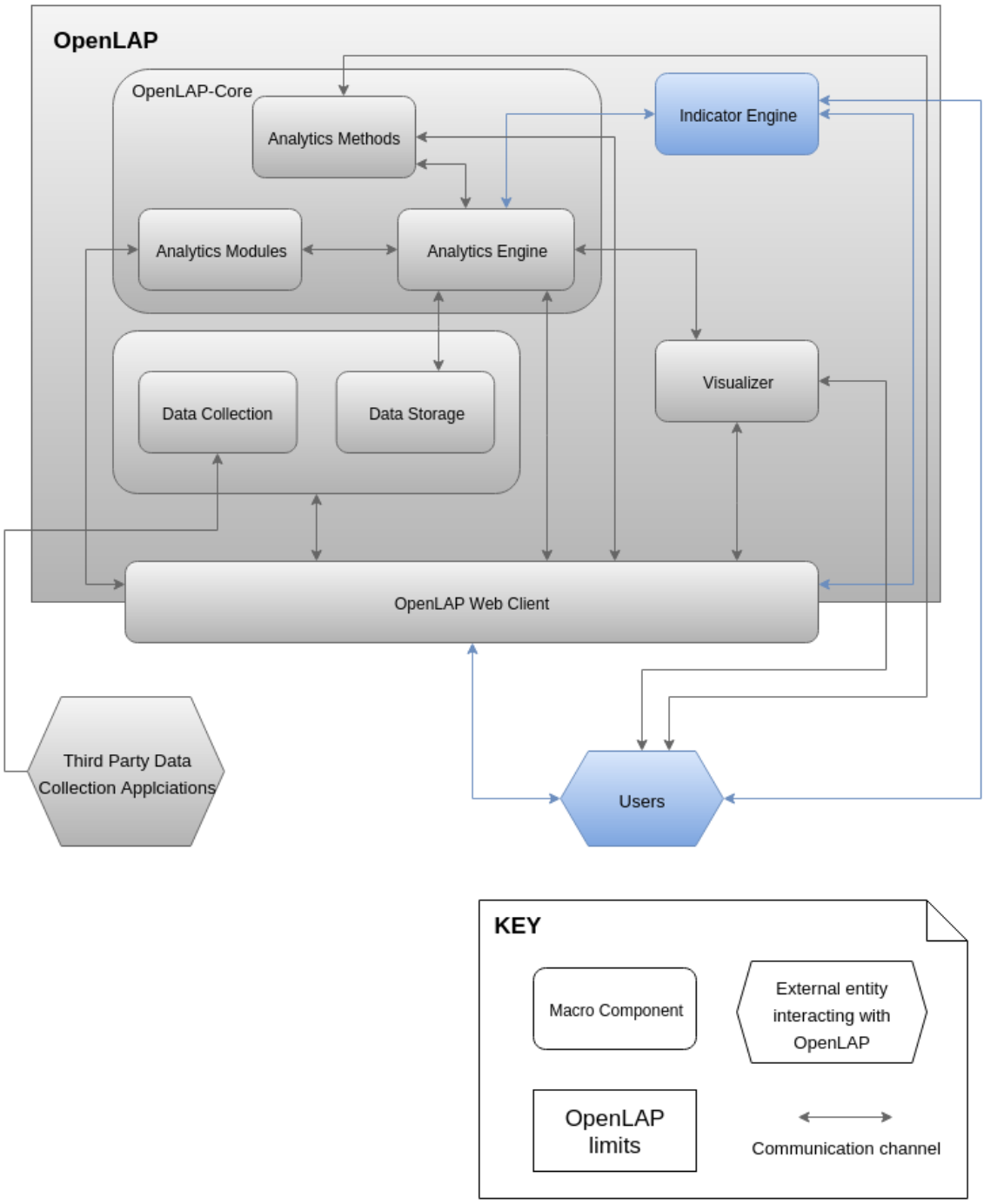
- **<<Interface>>GLACategoryDAO**: (There must exist a DAO for each one of the Learning Context Data Model classes used, hence the multiplicity in the diagram) This class and the other corresponding Learning Context Datamodel DAOs are in charge of handling the Data Access for the corresponding model objects described in the package `datamodel`. For each Class of the `datamodel` package and at least for each relevant class of the Learning Context Data Model there must exist a corresponding Data Access Layer Class interface DAO.
- **GLACategoryDaoImpl**: Implementaiton for each of the DAOs provided in the aforementioned item.
- **<<Interface>>GLAIndicatorDao**: Data access Object interface for the Indicators of the OpenLAP system.
- **GLAIndicatorDaoImpl**: Implementaiton for each of the DAOs provided in the aforementioned item.
- **<<Interface>>OperationsDao**: Interface for the Data Access Object handling Indicators in this macro component.
- **OperationsDaoImpl**: Implementation for the Data Access Object Interface for handling operations in the indicator engine.
- **<<Interface>>GLAQuestionDao**: Data access Object interface for the Questions of the OpenLAP system.

- **GLAQuestionDao**: Implementation for the Data Access Object Interface for the Questions of the OpenLAP system.

Package: datamodel

- **GLACategory**: (A class like this should be present for each relevant class of the Learning Context Data Model). This class represents the Category element of the Learning Context Datamodel.
- **GLAIndicator**: Data model object to represent the Indicators of the OpenLAP system.
- **GLAQuestion**: This class holds the relevant data for representing the Indicator Engine Questions.
- **GLAOperations**: This class holds the relevant data for representing the different Indicator operations types, for example: number, correlation, regression, etc. The operation is then turned into a set of queries through the indicator_system logic and those queries are saved into the indicator.
- **AnalyticsMethod**: This is a representation of the metadata required for the user to be able to isntantiate an indicator, that is, the user will associate an indicator to a determined Analytics Method in order to be able to produce post processing. This class holds the metadata necessary for the user to determine if they want to associate a determined Analytics Method to the indicator or not. Properties include description, name, availability, Indicator Operations it is available for and description of fields.
- **Visualization**: Similar to the previous class, this is a representation of the metadata of a determined visualization method that is made available from the Visualizer macro Component and used to instantiate an indicator (associating the indicator to an Analytics Method and a Visualization). The class holds metadata of the particular visualization (name, description, availability), what operations it is available for and a preview of data for the controller to be showed to the user and ease the instantiation of the indicator.

Context Diagram



Variability Guide

- User and session details have been omitted since they portray another aspect of the Indicator Engine. Ideally, the methods are to be restricted depending on the user role.
- Whenever new indicator rules are to be implemented, they must be connected to a consumer class in the indicator_system package within a sub-package. The class is in charge of interpreting the rules and providing a service to the controller, which needs to be then extended to hold the additional methods for interacting with any new rules and provide API endpoints in order to use them.
- The Indicator Engine, additionally to the operations it offers, communicates with the OpenLAP Web Client macro component that is able to connect to the other macro components that are specified through this architecture in the Data Collection and Data Storage and Analytics Engine macro component Views.

Rationale

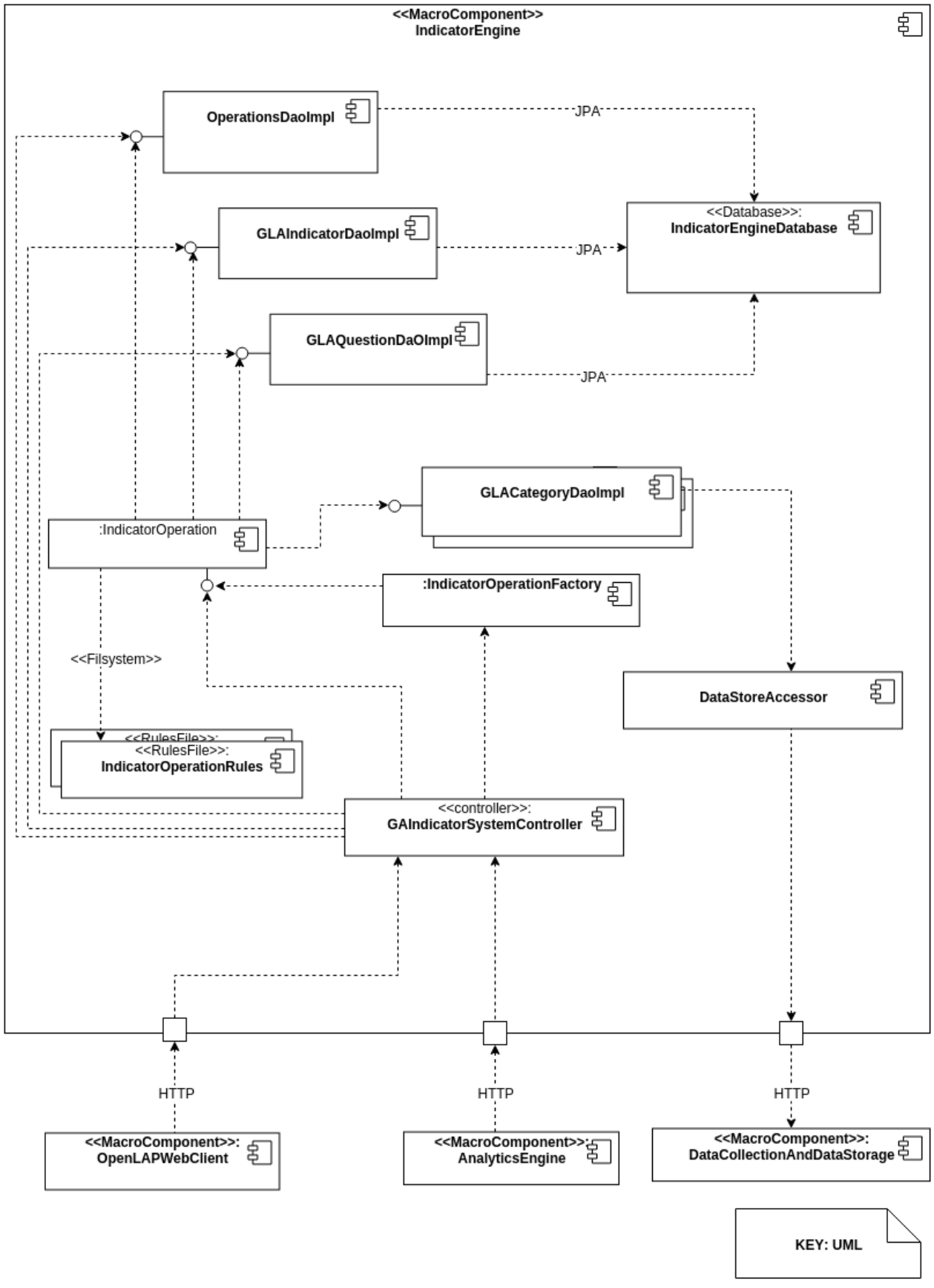
The Indicator Engine macro component main purpose is to provide personalization to the user so they can establish goals and utilize existing indicators to asses their goals or reuse existing indicators made by other users. The Indicator Engine also functions as an entry point to interact with the main usage of the platform: create the indicator request codes for embedding in other applications or the user's dashboard. Additionally, the Indicator Engine must provide a means to communicate with the OpenLAP Web Client macro component for user interaction.

Related Views

- Section 4.3.2. explains the C&C Views of the Indicator Engine
- Section 4.1. explains the Data Collection and Data Storage
- Section 4.7. explains the OpenLAP Web Client macro component.

INDICATOR ENGINE: COMPONENT AND CONNECTOR VIEWS

Primary Presentation



Element Catalog

Components

- **<<Controller>>:GAIndicatorSystemController**: The controller will serve as the main facade of the macro component. It exposes HTTP RESTful API so the Client(s) can interact with it. The Controller will interact mainly with the IndicatorOperationFactory when creating new Indicators that will provide it on Operations depending on what the client needs to process and allows to obtain the queries from the Indicator System (which is operated by each of the Indicator Operations available). When Indicators tat already exist are used, the queries that are already preestablished will be passed directly to the
- **IndicatorOperationFactory**: This component allows for decoupling the GAIndicatorSystemController and other components that need to access the IndicatorOperations that handle the rules system. It also allows for an extensible design that supports new operations when needed.
- **IndicatorOperation**: These classes will handle the use of the rules over the information that is passed from the controller. The Operation allows to establish types of Indicators and, if needed, consume a ruleset that is fed through a rule definition file.
- **<<RulesFile>>:IndicatorOperationRules**: This is a critical component of the Indicator Engine macro component, since it holds a rule system that transforms IndicatorOperations into SQL like rules to be passed to the Data Collection and Data Storage system based on the indicator that the user defines or selects.
- **GLACategoryDaoImpl**: An instance of a DAO implementation for each of the Learning Context Data Model must exist and be present in order for the Indicator Engine to access the data through the DataStoreAccessor and map it to the appropriate model entities.
- **DataStoreAccessor**: A proxy component that acts as a client from the present macro component to the Data Colection and Data Storage macro Component. It allows to access the API of the latter and acquire the data needed using the queries that the Indicators store depending on the operations.
- **AnaliticsMetodsAccessor**: Client for accessing the available Analytics Methods of the Analytics Methods macro component.
- **VisualizationAccessor**: Client for accessing the available Visualizations from the Visualization macro Component.
- **GLAQuestionDaOImpl**: A DAO object for accessing the stored Questions.
- **GLAIndicatorDaoImpl**: A DAO object for accessing the stored Indicators (and their queries).
- **OperationsDaoImpl**: A DAO object for accessing the stored Operations.
- **<<Database>>:IndicatorEngineDatabase**: This database stores all the Indicators, Operations and Questions of the OpenLAP Web Client system and is to be accessed directly as part of the Indicator Engine macro component. Whenever new Indicator are created, this is where they will be stored and accessed.
- **<<MacroComponent>>:OpenLAPWebClient**: The OpenLAP Web Client macro component.
- **<<MacroComponent>>>>:DataCollectionAndDataStorage**: The macro component described in section 4.1 Of this architecture.
- **<<MacroComponent>>>>:AnalyticsMethods**: The macro component described in section 4.5 Of this architecture.
- **<<MacroComponent>>>>:Visualizer**: The macro component described in section 4.4 Of this architecture.

Connections

- HTTP RESTful connections are to be established from the OpenLAP Web Client (and/or other clients) to the GAIndicatorSystemController. HTTP is also required to access other macro components. Internal Components such as the DataStoreAccessor act as a client-proxy to obtain data and other services exposed by the APIs of the different OpenLAP Web Client macro components. The AnalyticsEngine accesses the functionalities of the Indicator Engine through HTTP calls as well.
- JPA is ideally used between any Data Access Layer objects and the internal Databases, in this case the IndicatorEngineDatabase.
- Rules for the creation of Query Language strings for the querying of the Data Store are files that need to be provided along the macro component. If new rules are to be defined, they need to be created in their on package along with a class that extends the IndicatorOperation, which takes care of firing the rules on the settings passed by the GAIndicatorSystemController.
- The IndicatorOperationFactory is the object in charge of returning the appropriate IndicatorOperation object to the controller so it can access the pertinent rules through the object described in the previously.
- Other calls are standard procedure or remote procedure Java calls.

Variability Guide

- The main point of variation is when new packages with new Indicator Operations are introduced. The Controller will be configured so it can pass the required IndiatorOperation and the IndicatorOperationfactory will take care of providing the appropriate object to the controller, The controller will then feed the data entered by the user through the client and obtain a query. The Extension mechanism of IndicatorOperations is then were is possible to change statically defined operations in the system.

Rationale

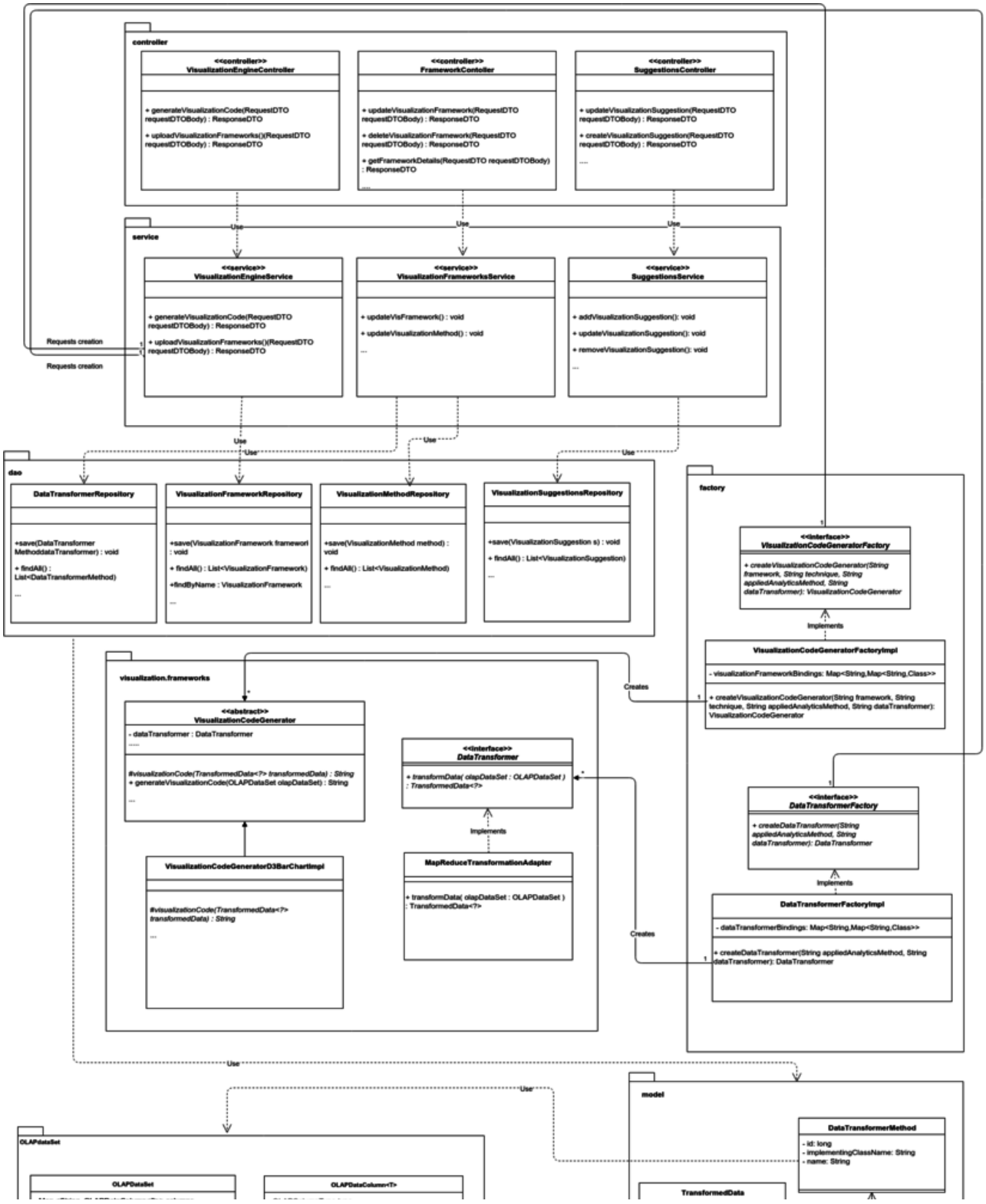
The main idea of this view of the macro component is to establish a mechanism for creating a rule system that transforms information entered through the controller into queries that can be later on used on the Data Collection and Data Storage macro component. Th data will be fetched by the Analytics Engine which will then use a configuration to pipe it to the appropriate Analytics Method and Visualization and be able to return an Indicator request code. Since the data corresponding to the Learning Context Data Model is stored in a different macro component the Analytics Engine must enable the connection. If necessary, the Indicator Engine has a client of its own in order to access the Data Collection and Data Store macro component.

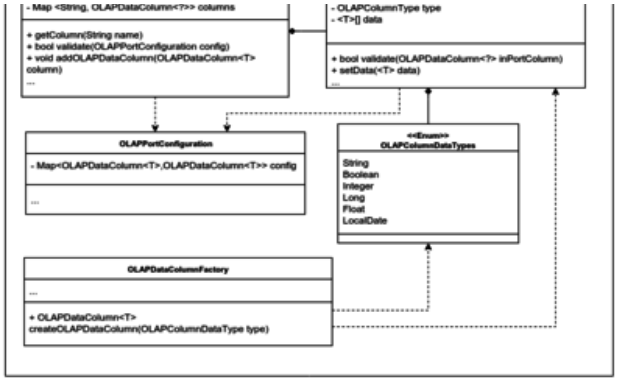
Related Views

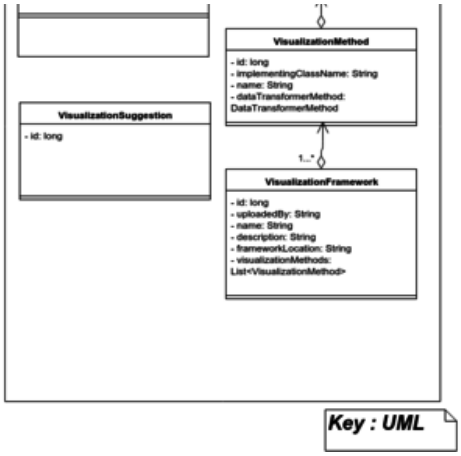
- Section 4.3.1 exposes the Module Views of this macro component
- Section 4.1 Exposes the overview of the Data Collection and Data Storage, where the queries will be eventually executed.
- Section 4.2 Exposes the overview of the Analytics Engine, where the queries, analysis and visualization is executed.
- Section 4.4 Exposes the overview of the Visualizer. The present macro component requires access to the available visualization libraries.
- Section 4.5 Exposes the overview of the Analytics Methods. The present macro component requires visibility of the available Analytics Methods.

VISUALIZER: MODULE VIEWS

Primary Presentation







Element Catalog

Package Controller

A controller package to hold the HTTP RESTful endpoints for the Visualization frameworks and methods upload. In addition, the controllers also provide an endpoint to get the client visualization code to be embedded in a Webpage. Furthermore, endpoints to manage the list of applicable visualizations for a certain data set (determined by static set of visualization rules) are also included. All endpoint can be accessed by client (Analytics Engine) by making an HTTP Request.

- **VisualizationEngineController**: This class exposes HTTP endpoint to get the generated client Visualization code. The clients are expected to provide the framework and method details to the endpoint in order to get the correct visualization code.
- **VisualizationFrameworksController**: This class exposes HTTP RESTful endpoints to perform CRUD operations on the Visualization Frameworks and their Methods. Researchers/Developers can work with existing Visualization Frameworks / Methods in the Visualizer as well as upload new ones whereas the normal users can simply View the existing ones.
- **VisualizationSuggestionsController**: This class exposes HTTP RESTful endpoints to perform CRUD operations on the Visualization Suggestions (rules which state which visualization method represents which OLAPDataset configuration).

Package Service

The package service provides an abstraction and contains the business logic classes. These classes provide methods to be used by the Controllers.

- **VisualizationEngineService**: This class contains methods which can be used to generate the client visualization code by providing the stored Visualization Framework id/name and its connected Visualization Method id/name.
- **VisualizationFrameworksService**: This class provides methods to search for existing Visualization Frameworks or Methods, as well as to update their attributes or remove them completely. Furthermore, the service also contains methods to handle the upload of Visualization Frameworks. These upload methods along with the bundled JAR file require configuration data (metadata) which acts as a manifest specifying exactly which frameworks, methods and data transformers are part of the file being uploaded.
- **VisualizationSuggestionService**: Service which provides methods retrieve and perform CRUD operations on the Visualization Suggestions. THe Visualization Suggestions are simple rules(entries) which connect an existing Visualization Method of a

Visualization Framework with an OLAPDatasetColumnConfiguration.

Pakage dao

The dao package contains the classes which serve as the data access layer for handling the metadata of Visualization Frameworks,Methods and Suggestions.

- **DataTransformerMethodsRepository**: An interface listing the methods for accessing the metadata of DataTransformers
- **VisualizationFrameworkRepository**: Provides a list of methods for accessing the metadata of Visualization Frameworks. Contains methods for retrieving, creation and removal of frameworks
- **VisualizationMethodRepository**: An interface listing the methods for performing CRUD operations on the metadata of Visualization Methods of a Visualization Framework

Package framework

The package framework contains the interfaces and abstract classes which are to be extended by new implementations of Visualization code generators and Data transformers

- **VisualizationCodeGenerator**: An abstract class which contains the control flow implementation of generating visualization code for the client. This class contains two abstract methods which each new Visualization code generator will have to override, namely, the "initializeDataSetConfiguration" method and the "visualizationCode" method. The developer when overriding the "initializeDataSetConfiguration" method should specify what kind of input (OLAPDataSet configuration) and output (OLAPDataSet configuration) are expected by the visualization code generator, it should be noted that at the moment the output can left null. The overriden method "visualizationCode" contains the actual client visualization code.
- **DataTransformer**: An interface containing list of methods which need to be implemented by each concrete DataTransformer. The DataTransformer method to transform data will be called by the VisualizationCodeGenerator to transform the OLAPDataSet data received from the client to a representation which is understood by the Visualization code generation method.

Package Model

Model package which holds all the model classes relevant to the Visualizer component.

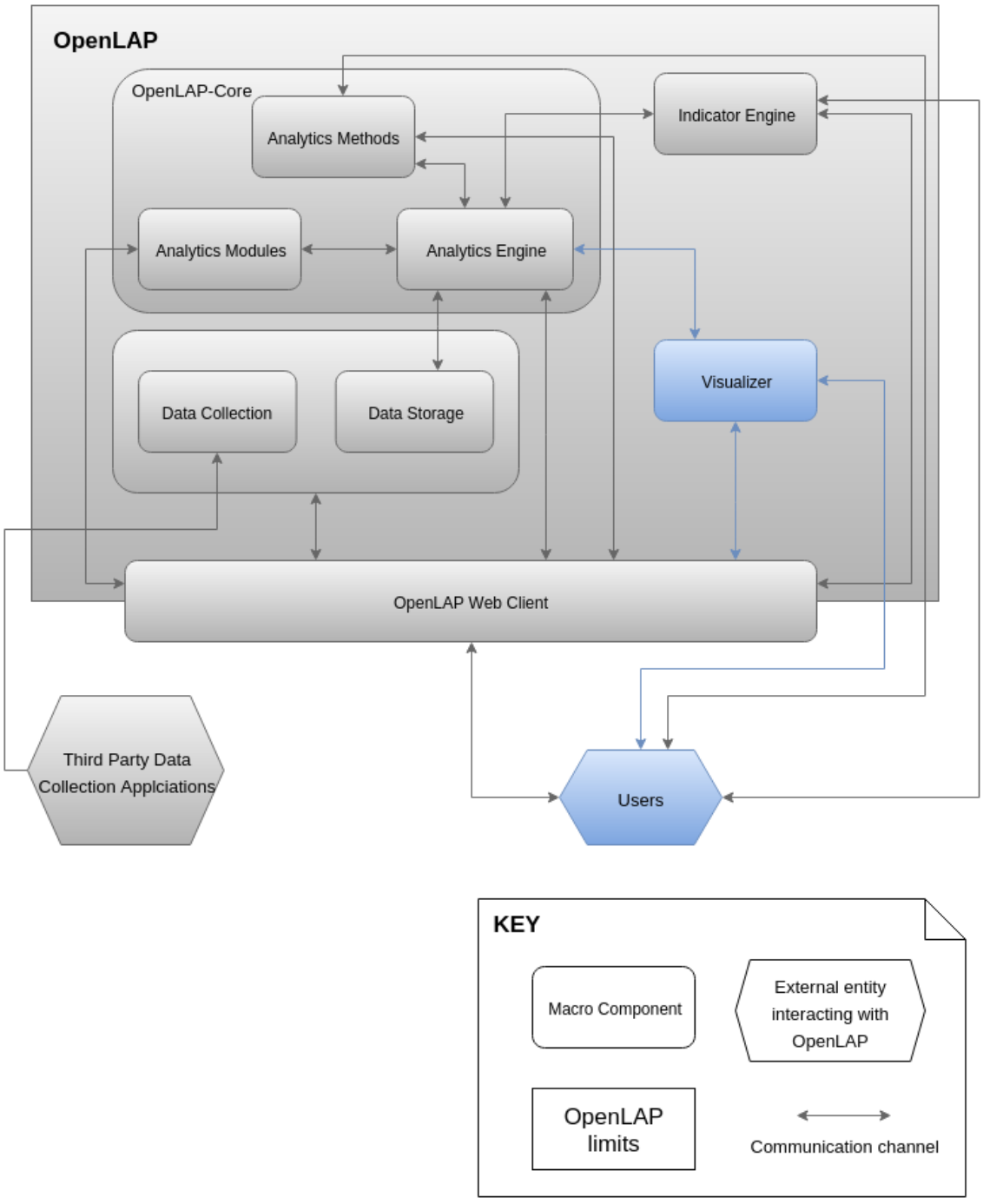
- **TransformedData**: Class which is envisioned to be a base class for polymorphic datatypes which represent the data after transformation is applied to it

The following three model classes represent the properties stored in the database and details the relations they have between each other. These classes are also used as models for serialization and deserialization to/from JSON.

- **VisualizationFramework**: Class representing the details of a Visualization Framework
- **VisualizationMethod**: Class representing the details of a Visualization Method belonging to a Visualization Framework
- **VisualizationSuggestion**: Class representing the details of a Visualization Suggestion. A Visualization Suggestion is a simple database entry containing a OLAPDatasetColumnConfiguration and its Visualization Method. These entries let the client receive

suggestions of Visualization Methods for a given OLAPDatasetColumnConfiguration

Context Diagram



Variability Guide

- The VisualizationCodeGenerator abstract class is meant to be extended by new code generators that Researchers / Developers create. These code generators will simply return the respective HTML+JS code as a String, to be sent to clients. In addition, the

code generators also need to provide the OLAPDataset that they expect as input. The OLAPDataset will just include the column configuration without the actual data. The concrete implementation of the VisualizationCodeGenerator can be submitted as a bundled JAR through the API exposed in the VisualizationFrameworksController

- The DataTransformer interface provides a skeleton of the methods that are meant to be overriden by concrete implementations which the Researchers / Developers create. These concrete implementations will act as adapters which will transform the data (input) which will be in form of the OLAPDataset to TransformedData. The TransformedData object can then be used by the various VisualizationCodeGenerator to generate the visualization code. Similar to the VisualizationCodeGenerators the concrete implementation of the data transformers(adapters) can be submitted as a bundled JAR through the API exposed in the VisualizationFrameworksController.

Rationale

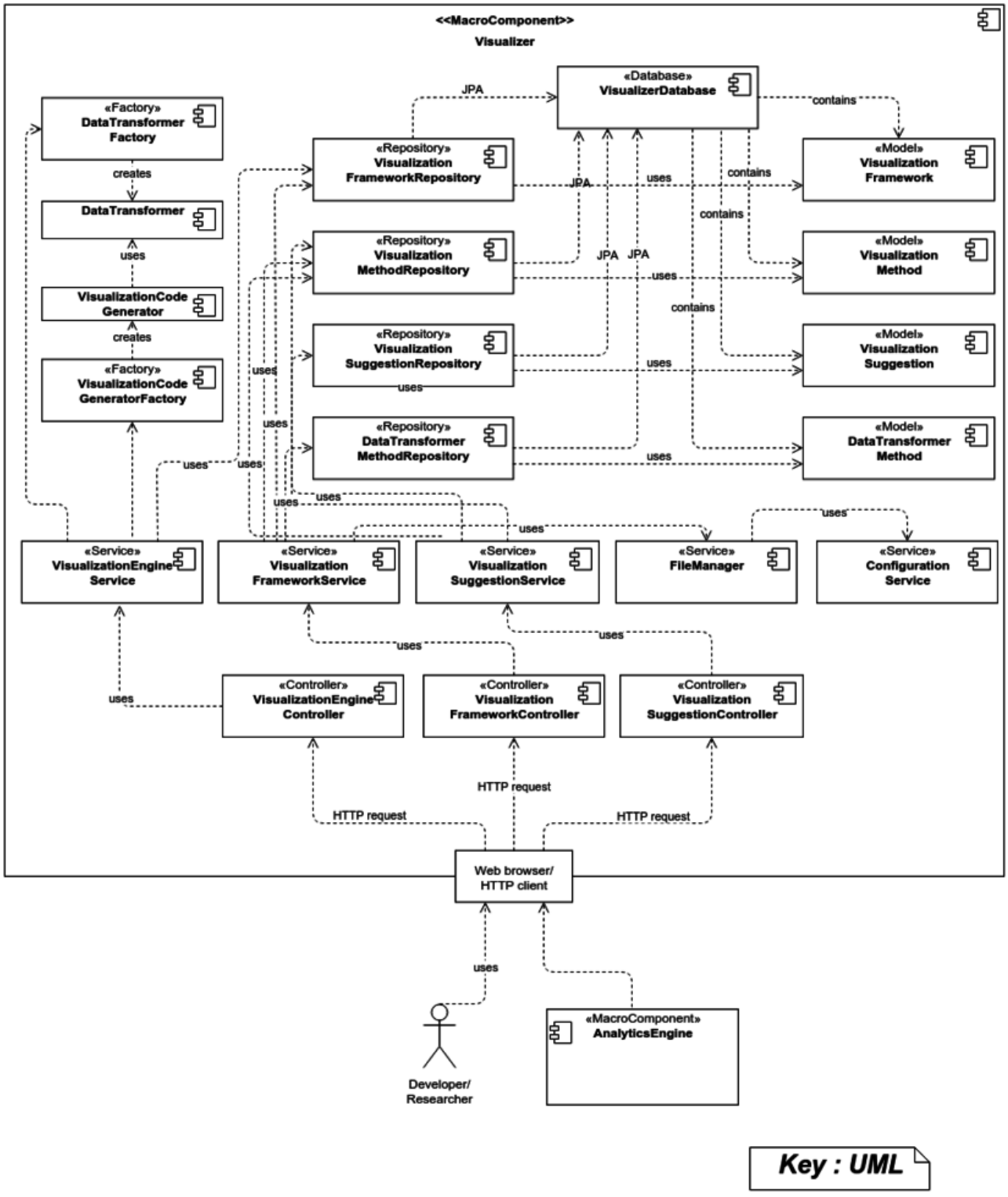
- The OLAPDataset (Column configuration) has to be included via an overriden method in each VisualizationCodeGenerator. Although, this would belong in the DataTransformer it was explicitly included in the VisualizationCodeGenerator due to the reason that a single DataTransformer can be reused by many VisualizationCodeGenerators.

Related Views

- Section 4.4.2 explains the C&C Views of the Visualizer
- Section 4.2 explains the Analytics Engine, which orchestrates the interaction with the Visualizer
- Section 4.5 explains the Analytics Methods, which process the data that is to be visualized
- Section 4.6 explains the Analytics Modules, which holds Triads that allow to save configurations between Indicators, Analytics Methods and Visualizations

VISUALIZER: COMPONENT AND CONNECTOR VIEWS

Primary Presentation



Element Catalog

Components

- **<<database>> VisualizerDataBase**: This is a database which holds the data containing the uploaded Visualization

Frameworks,Methods and Suggestions along with the Data Transformers.The CRUD operations are performed on the database via the appropriate repositories

- **<<repository>> VisualizationFrameworkRepository**: Provides an interface for database operations (creation, updating, retrieval or deletion of records) to be performed on the `VisualizationFramework`
- **<<repository>> VisualizationMethodRepository**: Provides an interface for interacting with the `VisualizationMethod`
- **<<repository>> VisualizationSuggestionRepository**: Provides an interface for database operations to be performed on the `VisualizationSuggestion`
- **<<repository>> DataTransformerRepository**: Provides an interface for interacting with the `DataTransformer`
- **<<model>>VisualizationFramework**: The object holds the information about a VisualizationFramework that was uploaded by Developers/Researchers to the Visualizer macrocomponent
- **<<model>>VisualizationMethod**: Instance of this component holds the information about a VisualizationMethod. A VisualizationMethod belongs to a VisualizationFramework and is part of the bundled JAR that was uploaded by Developers/Researchers to the Visualizer macrocomponent
- **<<model>>VisualizationSuggestion**: An instance of this component holds the information about a VisualizationFramework that was uploaded by Developers/Researchers to the Visualizer macrocomponent
- **<<model>>DataTransformerMethod**: The object holds information about a DataTransformer that was uploaded by Developers/Researchers to the Visualizer macrocomponent
- **<<factory>>DataTransformerFactory**: An abstract factory providing functions to create of DataTransformer objects at runtime. These objects contain the concrete behavior to transform data from the OLAPDataSet into a format parseable by the relative VisualizationMethod
- **<<factory>>VisualizationCodeGeneratorFactory**: An abstract factory providing functions to create VisualizationCodeGenerators (VisualizationMethod) objects at runtime. Instances of the VisualizationCodeGenerator contain client visualization code
- **<<factory>>DataTransformerFactory**: An abstract factory providing functions to create of DataTransformer objects at runtime
- **DataTransformer**: A concrete instance created by the `DataTransformerFactory`
- **VisualizationCodeGenerator**: A concrete instance created by the `VisualizationCodeGeneratorFactory`
- **<<service>>VisualizationEngineService**: A service which provides methods for the `VisualizationEngineController` to call to generate the visualization code
- **<<service>>VisualizationFrameworkService**: This is a core administration service of the `Visualizer` macrocomponent. It provides methods for the `VisualizationFrameworkController` to expose the available Visualization Frameworks and their Methods. In addition, also methods for the controller to create, update or delete existing ones
- **<<service>>VisualizationSuggestionService**: A service which exposes methods for the `VisualizationSuggestionController` to create new VisualizationSuggestion along with retrieving, updating and removing existing ones
- **<<service>>ConfigurationService**: A service which provides access to the configuration parameters of the application
- **<<service>>FileManager**: A service which exposes methods to access the file storage system of the `Visualizer` macrocomponent
- **<<controller>>VisualizationEngineController**: This controller exposes a RESTful API with methods for generating the visualization code
- **<<controller>>VisualizationFrameworkController**: This controller exposes a RESTful API with methods for uploading, modifying, removing and browsing `VisualizationFramework` and `VisualizationMethod`. It also contains an endpoint to check the input ports of a `VisualizationMethod`.
- **<<controller>>VisualizationSuggestionController**: This controller exposes a RESTful API with methods for uploading, modifying, removing and browsing `VisualizationSuggestion`
- **<<user>>Developer/Researcher**: The user interacts with the Visualizer macrocomponent via sending HTTP requests to the RESTful API exposed by the controllers
- **<<macrocomponent>>AnalyticsEngine**: Being the core macrocomponent of the OpenLAP platform the `AnalyticsEngine` has full access to the `Visualizer` API. It is through the `AnalyticsEngine` that user/client requests will reach the `Visualizer`

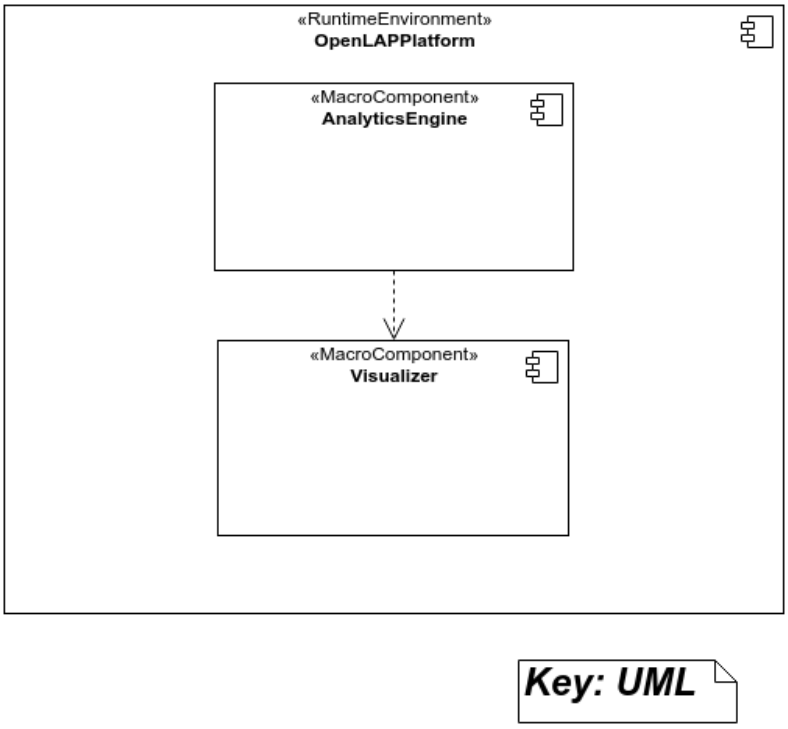
Connectors

- **HTTP request**: Endpoints of the controllers are accessed via HTTP requests sent by clients
- **JPA**: Java Persistence API specification used to access the database
- **VisualizerDatabase<<contains>>VisualizationFramework**: Represents records of `VisualizationFramework` stored in the `VisualizerDatabase`
- **VisualizerDatabase<<contains>>VisualizationMethod**: Represents records of `VisualizationMethod` stored in the `VisualizerDatabase`
- **VisualizerDatabase<<contains>>VisualizationSuggestion**: Represents records of `VisualizationSuggestion` stored in the `VisualizerDatabase`
- **VisualizerDatabase<<contains>>DataTransformerMethod**: Represents records of `DataTransformerMethod` stored in the `VisualizerDatabase`
- **VisualizationFrameworkRepository<<uses>>VisualizationFramework**: The repository `VisualizationFrameworkRepository` uses the model `VisualizationFramework` to map the objects to database records and vice versa
- **VisualizationMethodRepository<<uses>>VisualizationMethod**: The repository `VisualizationMethodRepository` uses the model `VisualizationMethod` to map the objects to database records and vice versa
- **VisualizationSuggestionRepository<<uses>>VisualizationSuggestion**: The repository `VisualizationSuggestionRepository` uses the model `VisualizationSuggestion` to map the objects to database records and vice versa
- **DataTransformerMethodRepository<<uses>>DataTransformerMethod**: The repository `DataTransformerMethodRepository` uses the model `DataTransformerMethod` to map the objects to database records and vice versa
- **VisualizationEngineService<<uses>>DataTransformerFactory**: `VisualizationEngineService` uses the factory methods provided by `DataTransformerFactory` to create instances of the required `DataTransformer` in order to transform data from the OLAPDataSet into a form parseable by the `VisualizationMethod`
- **VisualizationEngineService<<uses>>VisualizationCodeGeneratorFactory**: `VisualizationEngineService` uses the factory methods provided by `VisualizationCodeGeneratorFactory` to create instances of the required `VisualizationCodeGenerator` in order to generate the client visualization code
- **VisualizationEngineService<<uses>>VisualizationFrameworkRepository**: `VisualizationEngineService` uses the repository `VisualizationFrameworkRepository` to get access to the `VisualizationFramework` information existing in the database. From this information the service can create the relevant instances required to generate the visualization code
- **VisualizationFrameworkService<<uses>>VisualizationFrameworkRepository**: `VisualizationFrameworkService` uses the repository `VisualizationFrameworkRepository` to perform CRUD operations on the `VisualizationFramework` records
- **VisualizationFrameworkService<<uses>>VisualizationMethodRepository**: `VisualizationFrameworkService` uses the repository `VisualizationMethodRepository` to perform CRUD operations on the `VisualizationMethod` records
- **VisualizationFrameworkService<<uses>>DataTransformerMethodRepository**: `VisualizationFrameworkService` uses the repository `VisualizationMethodRepository` to perform CRUD operations on the `DataTransformerMethod` records
- **VisualizationFrameworkService<<uses>>FileManager**: `VisualizationFrameworkService` uses the the `FileManager` service to perform file based operations on the Visualizer macrocomponent's storage system
- **VisualizationSuggestionService<<uses>>VisualizationSuggestionRepository**: `VisualizationSuggestionService` uses the repository `VisualizationSuggestionRepository` to perform CRUD operations on the `VisualizationSuggestion` records in the database
- **VisualizationSuggestionService<<uses>>VisualizationMethodRepository**: `VisualizationFrameworkService`

uses the repository `VisualizationMethodRepository` to retrieve information about `VisualizationMethod` records in order to make the connection between a `VisualizationMethod` and a `VisualizationSuggestion`

- **FileManager<<uses>>ConfigurationService**: `FileManager` uses the `ConfigurationService` to access to the application's configuration parameters. These configuration parameters include the temporary and permanent locations where the `FileManager` should store files along with the extension of the uploaded files
- **VisualizationEngineController<<uses>>VisualizationEngineService**: `VisualizationEngineController` uses the `VisualizationEngineService` to service the incoming client requests. Client request reaching the `VisualizationEngineController` include that for generation of visualization code using a certain `VisualizationFramework` and `VisualizationMethod`
- **VisualizationFrameworkController<<uses>>VisualizationFrameworkService**: `VisualizationFrameworkController` uses the `VisualizationFrameworkService` to service the incoming client requests. These clients requests range from performing CRUD operations on the `VisualizationFramework`, `VisualizationMethod` and `DataTransformer` to validating a `VisualizationMethod`'s input ports
- **VisualizationSuggestionController<<uses>>VisualizationSuggestionService**: `VisualizationSuggestionController` uses the `VisualizationSuggestionService` to service the incoming client requests. These clients requests include performing CRUD operations on the `VisualizationSuggestion` records in the database

Context Diagram



Variability Guide

- <<Factory>>DataTransformerFactory = The <<Factory>>DataTransformerFactory is an interface which has to be implemented to provide a concrete implementation of the factory. The methods which are implemented contain the code to create instances of the `DataTransformer` from the class name. The default implementation of the factory uses the JCL library to load the

`DataTransformer` classes

- <<Factory>>VisualizationCodeGeneratorFactory = The <<Factory>>VisualizationCodeGeneratorFactory is an interface which has to be implemented to provide a concrete implementation of the factory. The methods which are implemented contain the code to create instances of the `VisualizationCodeGenerator` from the class name. The default implementation of the factory uses the JCL library to load the `VisualizationCodeGenerator` classes
- VisualizationCodeGenerator = The `VisualizationCodeGenerator` is an abstract class which has methods that must be overriden by the concrete implementations. These abstract methods specify the generation of the visualization code which is sent back to client requests as well as the input and output ports in the form of the OLAPDataSet. The output ports can be safely ignored, however the input port configuration is crucial as it is used to determine if the incoming OLAPDataSet can be processed by the `VisualizationCodeGenerator`
- DataTransformer = The `DataTransformer` is an interface which has to be implemented to provide for a concrete implementation of the algorithm to transform data from the OLAPDataSet into a form parseable by the `VisualizationCodeGenerator`

Rationale

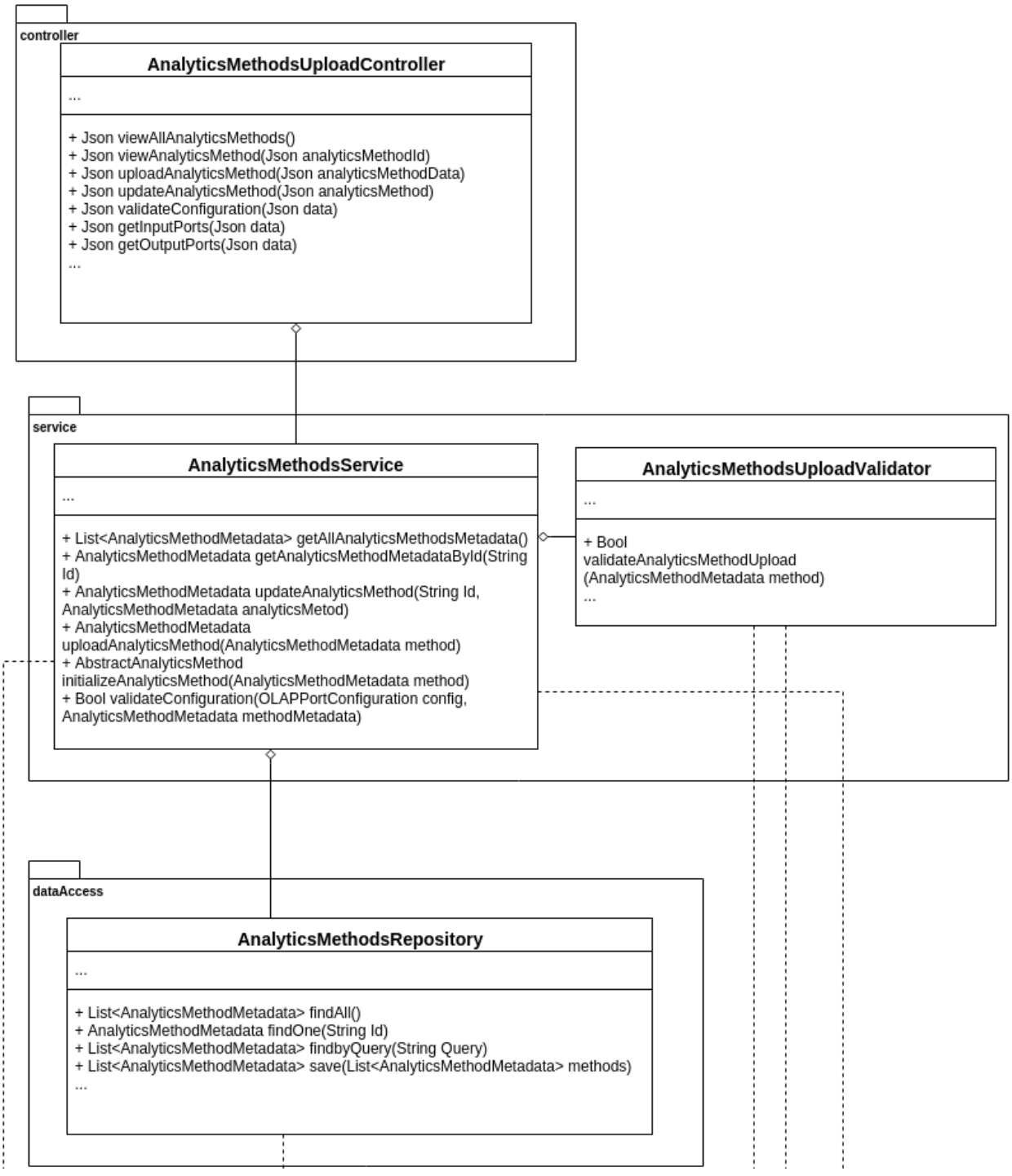
- The OLAPDataset (Column configuration) has to be included via an overriden method in each `VisualizationCodeGenerator`. Although, this would belong in the `DataTransformer` it was explicitly included in the `VisualizationCodeGenerator` due to the reason that a single `DataTransformer` can be reused by many `VisualizationCodeGenerators`

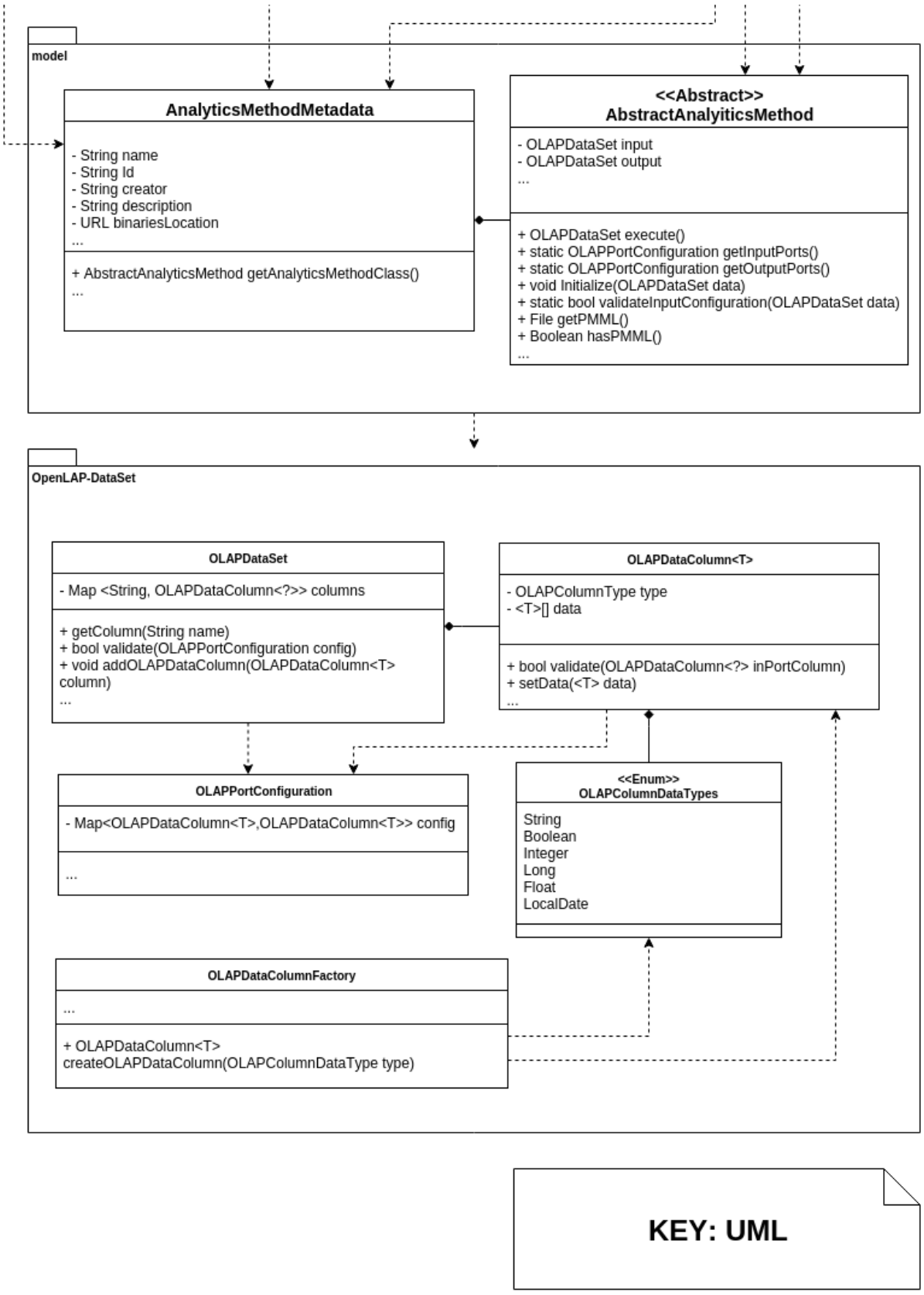
Related Views

- Section 4.4.1 explains the static Module Views of the Visualizer
- Section 4.2 explains the Analytics Engine, which orchestrates the interaction with the Visualizer
- Section 4.5 explains the Analytics Methods, which process the data that is to be visualized
- Section 4.6 explains the Analytics Modules, which holds Triads that allow to save configurations between Indicators, Analytics Methods and Visualizations

ANALYTICS METHODS: MODULE VIEWS

Primary Presentation





Element Catalog

Package controller

A controller package to hold the HTTP Restful endpoints for the Analytics Methods upload and modification.

- **AnalyticsMethodsUploadController**: This class will expose HTTP endpoints in a RESTFUL manner to upload, modify, and view Analytics Methods (specifically its metadata) as well as methods for retrieving configuration objects and validation of input. The upload, view and modification methods will be available to Researchers and Developers, while the view methods are available to other Users. The Analytics Engine will be able to directly access the Analytic Methods through another class.

Package service

This package serves for giving logical services when uploading and modifying Analytics Methods. It allows for validation, checking input/output and listing. It also enables the Analytics Engine to instantiate Analytics Methods.

- **AnalyticsMethodsService**: This class provides services to upload, modify and view the AnalyticsMethodMetadata of the different methods. Allows for establishing a link with the AnalyticsMethodRepository and gives access to the Analytics Engine to the specific AbstractAnalyticsMethod instances (the actual implementation submitted by the developers and researchers). This is possible due to the fact that the Analytics Methods reside on the same Java Virtual Machine as the Analytics Engine.
- **AnalyticsMethodsValidator**: When the Researcher/Developer uploads a new Analytics Method, the uploaded Jar file is sent along a JSON object metadata about the Analytics Method as well as a class that extends the AbstractAnalyticsMethod. Additionally, when the Analytics Method uploaded is a Predictive Model, it must provide a PMML file (through the respective method of the Analytics Method) to enable the standardization of the method technique. This validator helper class allows to perform checks that there exists an appropriate AbstractAnalyticsMethod extension class and that the PMML is a valid file. It is used by the Service when an upload or modification of an Analytics Method occurs.

Package dataAccess

A Data Access Layer for handling the Metadata of the Analytics Methods.

- **AnalyticsMethodsRepository**: This Repository enables a Data Access Layer for accessing the stored AnalyticsMethodMetadata necessary to locate and use the Analytics Methods.

Package model

The main classes that represent the usage of the Analytics Methods framework. Provides classes for locate and hold metadata as well as classes for the actual implementation of the Analytics Methods.

- **AnalyticsMethodMetadata**: This class has the required data needed for the Controller to expose the main data of the Analytics Method such as name, creator, version, description and also provides a locator for the Service to instantiate the Implementation of the specific AbstractAnalyticsMethod that is used by the Analytics Engine. This data is stored in the database whenever an upload or modification is made over the Analytics Method into a Database through the Repository.
- **AnalyticsMethodInstance**: This is the class that Researchers and Developers must extend in order to create specific Analytics Methods. It has an `execute()` method that will be executed by the Analytics Engine. It also has methods to expose the input and output configuration (`getInutPorts()` and `getOutputPorts()`) and a method to instantiate the data of is input.

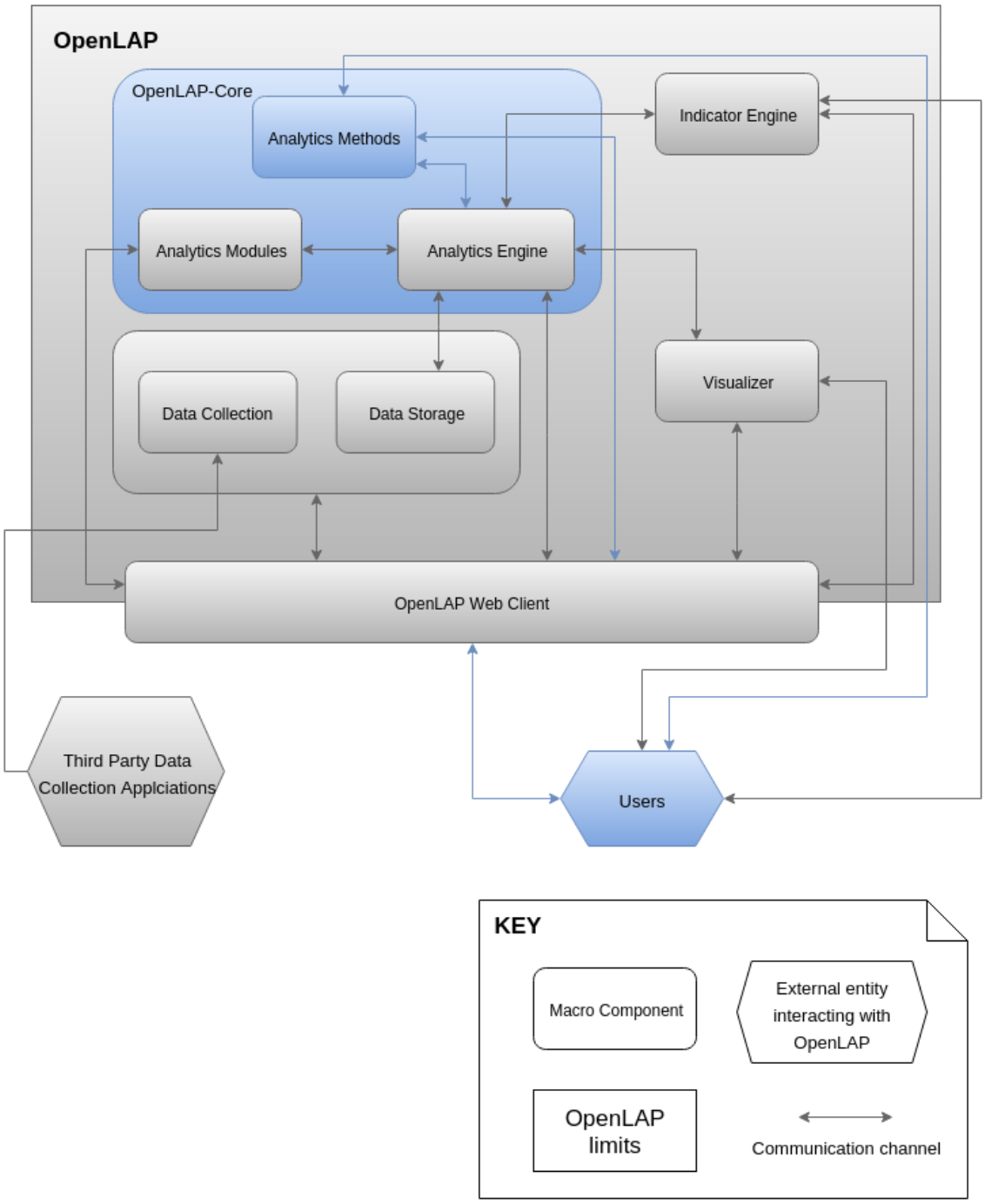
This class must also implement a `validateInputConnfiguration(OLAPDataSet config)` in order to dynamically check that the configuration that is getting i a valid mapping of input to its ports. The class uses heavily the `OLAPdataSet` package that is common among the components that reside within the Java Virtual Machine that contains the Analytics Engine (Analytics Engine, Analytics Methods and Visualization Engine). The reason for this is to establish a common simple Data Object that can be translated into columns and easily serializable into JSON.

Package OpenLAP-DataSet

This is a common package for the components that reside on the same JVM as the Analytics Engine. It allows the different macro components in it to have a common data object that is generic and compatible with table-like data. The model is simple but effective and requires minimum effort from researchers and developers to use.

- **OLAPDataSet**: This is a container for columns. It encapsulates a map between column names and the `OLAPDataColumn`s. When Researchers and Developers are creating new Analytics Methods or Visualizations they must use this data transfer object in order to allows dynamic type checking from the sources. The possible mappings are (Data from Indicator queries to Analytics Methods or Visualizations
- **OLAPDataColumn<T>**: This class represents a one dimensional column that is to be aggregated by the `OLAPDataSet`. It holds properties for type and array of the data. It also allows to verify that an input is valid for the specific column.
- **OLAPPortConfiguration**: This is a simple mapping between different columns. Its purpose is to support a dynamic checking for the input data from origin to the input port of the using classes. Different configurations can be: Input data form Indicators (Queries) to Analytics Methods or to Visualizations, Analytics Methods to Visualizations, Analytics Methods to other Analytics Methods.
- **OLAPDataColumnFactory**: A factory to create objects of type `OLAPDataColumn` depending on the types available on `<<Enum>>OLAPColumnDataTypes`.
- **<<Enum>>OLAPColumnDataTypes**: An enumerator to be used by the `OLAPDataColumnFactory`. Each entry represents valid column types.

Context Diagram



Variability Guide

- The `<<Abstract>>AbstractAnalyiticsMethod` is meant to be extended by new Analytics Methods that Researchers / Developers create. They will submit the code through the API exposed in the `AnalyticsMethodsUploadController` and the Analytics Engine will be able to execute the methods through the abstraction class.
- The classes described in the `OLAPdataSet` are meant to be used as data objects in the classes extending the `<<Abstract>>AbstractAnalyiticsMethod`. They will expose the data objects for configuration and validation through the Service (and transitively to the controller).

Rationale

This macro component is designed to reside in the same runtime environment as the Analytics Engine, i.e. the same Java Virtual Machine. This allows not only quick direct access to the Analytics Methods whenever a request for processing is made, but also enables the possibility to easily add new Analytics Methods that comply with the Modularity Framework to be added in runtime and be handled by the Framework dynamically. Users that are Researchers or Developers are able to upload new Analytics Methods via the endpoints exposed by the controller. This API enables them to plug in new techniques for analysis and submit them to the platform, which executes validation of the submitted work for compliance and then exposes the new Methods internally to make them available to the Analytics Engine. The submitted Analytics Methods are sent through Jar Files into the platform via the HTTP ports, where they are tested for compliance by the validator before made available.

The model of the platform is meant to hold abstractions for the Analytical Methods implementations. Users that are Researchers/Developers will extend the `<<Abstract>>AbstractAnalyticsMethod` class and implement the different methods and establish communication options for them to be accesses through the service to both the Controller for exposing the availability of methods, input and output validation and to the Analytics Engine to create instances and run the specific configurations. The data validation is performed through the use of the `OpenLAP-DataSet` package classes and allows to expose inputs as well as perform dynamic validation of the configurations.

Another key components is the `OpenLAP-DataSet` package, which is available to all the macro modules of the Analytics Engine runtime (also referred as the OpenLAP-Core) and enables the different components in it to transfer data in a two dimensional form. The reasons for establishing this data object are the following:

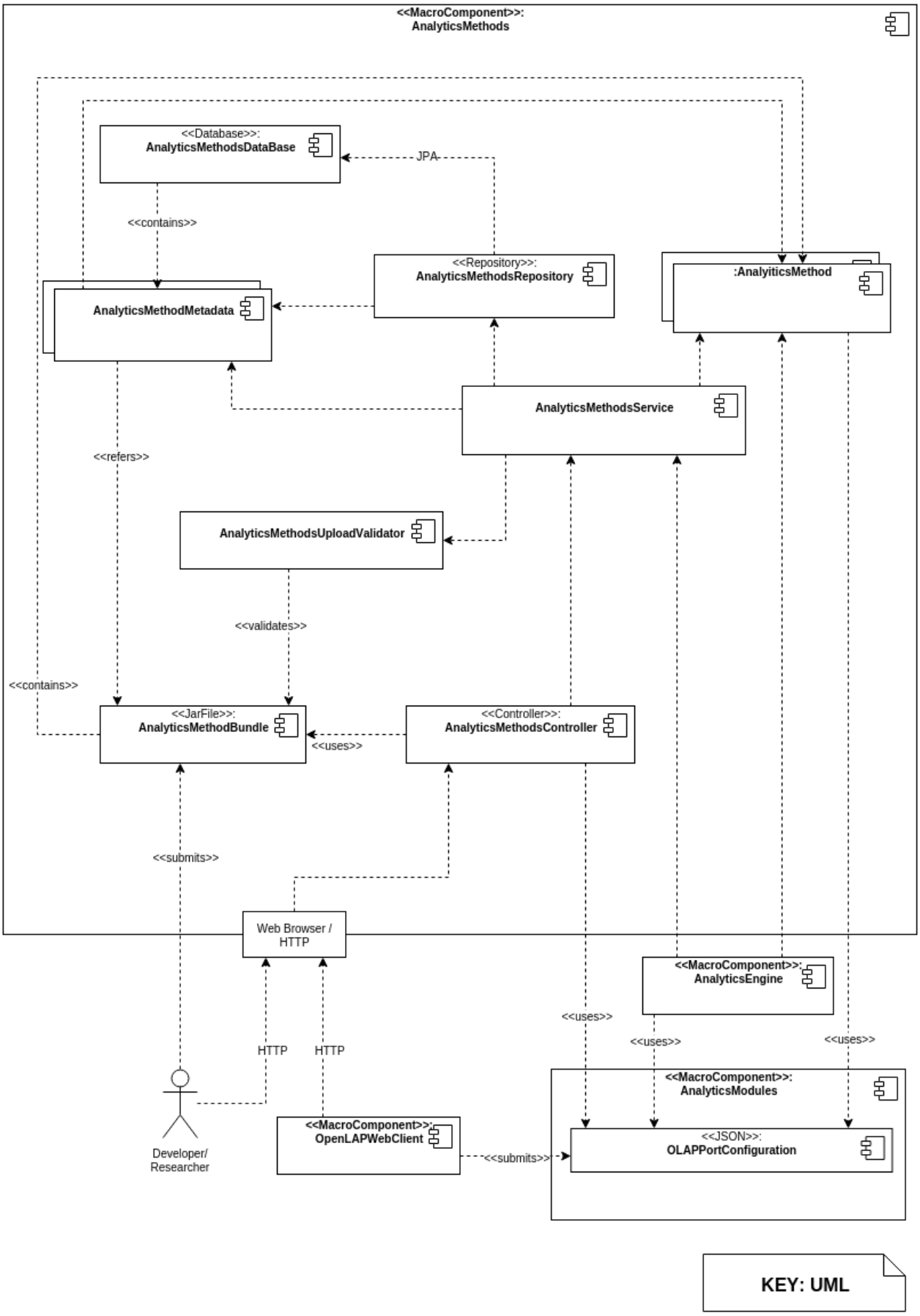
- An object for dynamic type checking is necessary since the outputs of Indicators and Analytics Methods must match the Inputs of other Analytics Methods and/or Visualizations.
- It is expected to be easy to implement by Developers and Researchers. and the guidelines for using when crating new Analytics Methods and/or Visualizations should be homogeneous and simple.
- Enables simple extension and allows Researchers/Developers to have flexibility on handling data internally in the Independent Analytics Methods / Visualizations.

Related Views

- Section 4.5.2. describes the Component and Connector view in order to understand the overview of the dynamic behavior of the component.
- Section 4.2. describes the Analytics Engine, the main macro component that interacts and uses the Analytics Methods. It also uses the `OpenLAP-DataSet` in order to pass data to the Visualizer macro component and/or to concatenate Analytics Methods.
- Section 4.4. describes the Visualizer, a macro component that also interacts with the `OpenLAP-DataSet` package.
- Section 4.6 describes the Analytics Modules, which holds Triads that allow to save configurations between Indicators, Analytics Methods and Visualizations

ANALYTICS METHODS: COMPONENT AND CONNECTOR VIEWS

Primary Presentation


<<MacroComponent>>:
AnalyticsMethods
<<Database>>:
AnalyticsMethodsDataBase
JPA
<<contains>>
<<Repository>>:
AnalyticsMethodsRepository
:AnalyiticsMethod
AnalyticsMethodMetadata
AnalyticsMethodsService
<<refers>>
AnalyticsMethodsUploadValidator
<<validates>>
<<contains>>
<<JarFile>>:
AnalyticsMethodBundle
<<uses>>
<<Controller>>:
AnalyticsMethodsController
<<submits>>
Web Browser /
HTTP
<<MacroComponent>>:
AnalyticsEngine
<<uses>>
<<uses>>
<<uses>>
HTTP
HTTP
<<MacroComponent>>:
AnalyticsModules
<<JSON>>:
OLAPPortConfiguration
Developer/
Researcher
<<MacroComponent>>:
OpenLAPWebClient
<<submits>>
KEY: UML

Element Catalog

Components

- **<<database>> AnalyticsMethodsDataBase**: This is a database which holds the metadata associated to the uploaded Analytics Methods. The record get updated whenever the Analytics Methods are modified or deleted and is accessed through the `AnalyticsMethodRepository`.
- **<<repository>> AnalyticsMethodsRepository**: Whenever the Analytics Methods are updated or uploaded, the respective metadata is inserted to the `AnalyticsMethodsDataBase` through this repository. Other interaction with the database is also handled by this object.
- **AnalyticsMethodMetadata**: This object holds the metadata of the different uploaded Analytics Methods that the Developers and Researchers submit to the system. Is used to locate the executables as well as provide relevant information that gets exposed in the `AnalyticsMethodsController`.
- **AnalyticsMethodsService**: The main service of the macro module. It provides both services for the controller to expose the available Analytics Methods as well as submit new methods. Additionally enables the Analytics Engine macro module to access the Analytics Modules to use and get instances for use.
- **AnalyticsMethodsUploadValidator**: Whenever new Analytics Methods are submitted, the validator takes care of accessing the submitted `AnalyticsMethodBundle` jar file and checks that it includes an extension of the `<<Abstract>>AbstracAnalyticsMethod` as well as valid JSON description metadata and PMML files (if needed).
- **<<jarFile>> AnalyticsMethodBundle**: This is a JAR file that holds the binaries of the Analytics Method. It contains at least a class that extends the `<<Abstract>>AbstractAnalyiticsMethod` Class and is uploaded along with a JSON object that describes the metadata. The Analyitcs Method can provide a PMML file that can be checked by the validator. Additionally the Jar file can contain other needed classes for the internal implementation of the method. These components are checked for validity by the `AnalyticsMethodsUploadValidator` during submission and/or modification. The `AnalyticsMethodsService` then provides an API for the Analytics Engine macro component to interact with instances of the Analytics Method submitted in the file.
- **<<controller>> AnalyticsMethodsController**: This controller exposes a RESTful HTTP API with methods for uploading, modifying, and browsing Analytics Methods. It also exposes methods for checking the input ports of available Analytics Methods as well as their metadata. Ideally, the `OpenLAPWebClient` will provide a we interface for this controller, but the methods can be used with direct HTTP requests.
- **<<user>> Developer/Researcher**: The Developer/Researcher interacts with the Analytics Methods upload features through HTTP. The Methods can be used through he instantiation of indicators with the help of the Analyitcs Engine macro component.
- **<<macroComponent>> AnalyticsEngine**: The Analytics Engine can access instances of the Analytics Methods (classes extending of `<<Abstract>>AbstracAnalyticsMethod`) through the usage of the `AnalyticsMethodsService`.
- **<<macroComponent>> AnalyticsModules**: The Analytics Modules macro component stores the configuration of mappings between the outputs of Indicators, Analytics Methods and/or Visualizations in the Triads. This configuration is saved when the indicator is created and is used directly by the Analytics Engine whenever requests for that specific indicator are executed. The configuration is then used over instances of the Analytics Methods in order to analyze data form the Indicators.

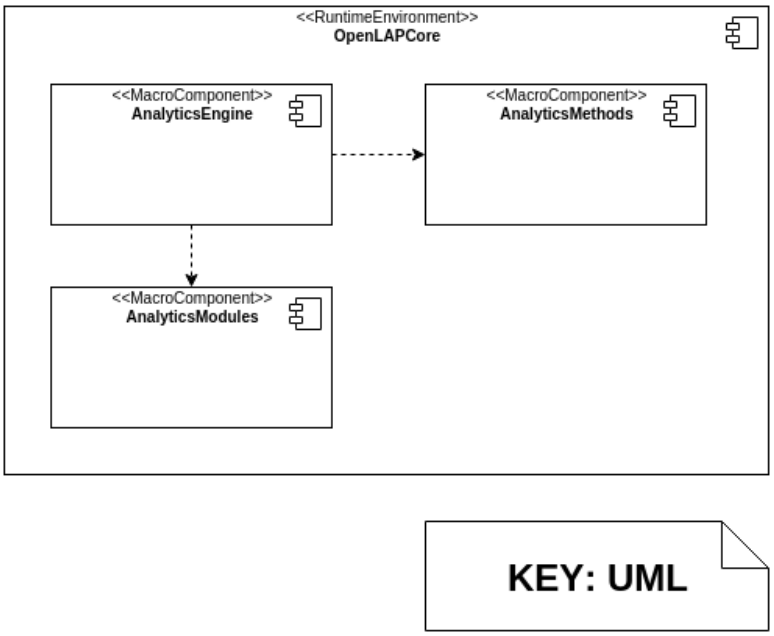
Connectors

- **HTTP**: Methods for the controller are accessed via HTTP or a browser. The controller exposes HTTP endpoints in order to support the design.
- **JPA**: JPA is used to access the Database.
- **AnalyticsMethodsDatabase <<contains>> AnalyticsMethodMetadata**: The stored items of the

`AnalyticsMethodsDataBase` are mainly mappings of `AnalyticsMethodMetadata`.

- **AnalyticsMethodMetadata <<refers>> AnalyticsMethodBundle**: The `AnalyticsMethodMetadata` contains the reference and data about the `AnalyticsMethodBundle`, in particular description fields and location.
- **AnalyticsMethodBundle <<contains>> :AbstractAnalyticsMethod**: The bundles have the Instrances (classes that extend the `<<Abstract>>AbstractAnalyiticsMethod`) of the Analytics Methods.
- **:AbstractAnalyticsMethod <<contains>> OLAPPortConfiguration**: The Analytics Method instances must be able to process OLAPPortConfiguration for the Triads with and check that the output of the Indicator query is compatible with its own input.
- **AnalyticsMethodsUploadValidator <<validates>> AnalyticsMethodBundle**: The validator checks that the Bundle is compilant with the framework's minimum guidelines.
- **AnalyticsMethodsController <<uses>> AnalyticsMethodBundle**: The Bundle is submitted to the Controller via HTTP.
- **AnalyticsMethodsController <<uses>> OLAPPortConfiguration**: The controller uses the OLAPPortConfiguration of the Triad in order to dynamically check that the types of input/output are correct with the help of the instances of the Analytics Methods.
- **AnalyticsEngine <<uses>> OLAPPortConfiguration**: The Analytics Engine accesses the OLAPPortConfiguration stored in the Triads.
- **Developer/Researcher <<submits>> AnalyticsMethodBunle**: The Developer/Researcher submits the bundle Jar file through HTTP.
- **OpenLAPWebClient <<submits>> OLAPPortConfiguration**: The Client can interact with the endpoits exposed by the controller sending requests to process OLAPPortConfiguration for validation.

Context Diagram



Variability Guide

- The OLAPPortConfiguration are specified according to the specific implementaitons of the classes that extend the `<<Abstract>>AbstractAnalyiticsMethod`. The OLAPPortConfiguration denote both the input and the output of the specific Analytics Method. The input will be provided by the Analytics Engine and is the result of executing the query that

the Indicator provides. The outputs will be sent to the Visualization specified on the Triad.

- The `<<Abstract>>AbstractAnalyiticsMethod` has methods that must be overriden by the implementations. The methods specify the implementation of the algorithm for execution as well as methods to inform the existence of and provide the PMML file if the implementer wishes to validate it. The implementations must also specify an `initialize(OLAPDataSet incomingData, OLAPPortConfiguration configuration)` method to initialize the inputs and forcing the implementation to use the passed configuration. The result of the execution of the main algorithm is stored on the Analytics Method output.

Rationale

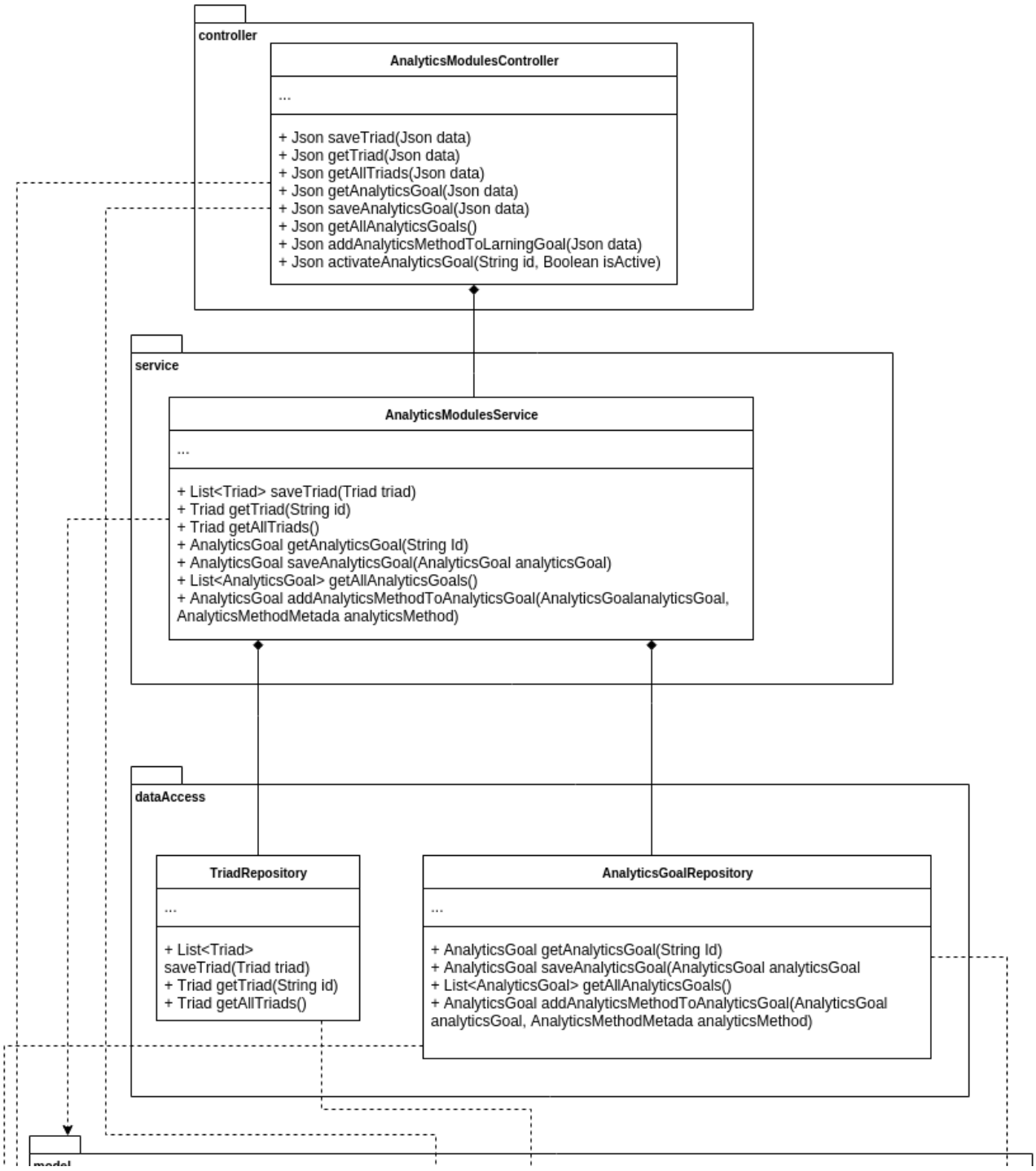
The Analytics Methods is a component that resides along the same Java Virtual Machine as the Analytics Engine. The reason behind this is that it allows a module to take care of the management of the Analytics Methods individually but at the same time gives direct access to the uploaded methods to the Analytics Engine. The idea is that the Analytics Engine can access the Analytics Modules in a dynamic fashion through the service. The users can submit new Analytics Methods by implementing the framework and uploading them trough the endpoints. Then, the runtime canexecute the Analytis Methods once they are deemed correct for functioning. The framework guarantees minimum execution context in order to do so. Additionally, the framework provides means for type checking of the configurations that are submitted through the controller and then stored in the Analytics Modules macro component inside the respective Triads. This enables the Analytics Modules to run against configurations that can be checked at runtime and be decoupled from the specific data that is used. The data about the Amalytics Method is submitted and stored within metadata objects that are stored whenever the method is uploaded or modified as well as holding references for the executable files (JARs) so the Service can relocate the jars if their location is changed.

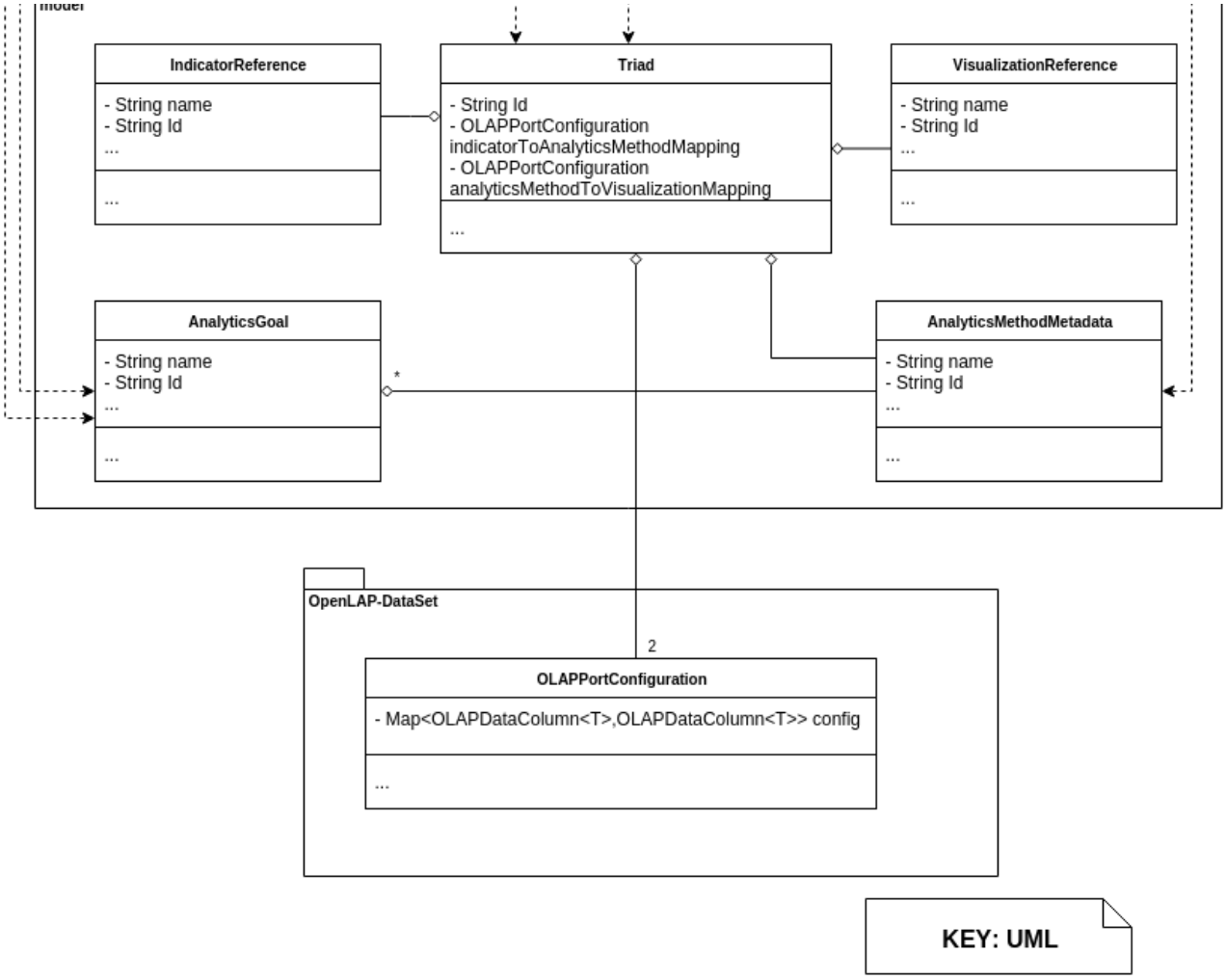
Related Views

- Section 4.5.1. describes the Modules view in order to understand the overview of the static behavior of the component.
- Section 4.2. describes the Analytics Engine, the main macro component that interacts and uses the Analytics Methods. It also uses the `OLAPdataSet` in order to pass data to the Visualizer macro component and/or to concatenate Analytics Methods.
- Section 4.4. describes the Visualizer, a macro component that also interacts with the `OLAPdataSet` package.
- Section 4.6. describes the Analytics Modules, which holds Triads that allow to save configurations between Indicators, Analytics Methods and Visualizations

ANALYTICS MODULES: MODULE VIEWS

Primary Presentation





Element Catalog

Package: controller

- **AnalyticsModulesController:** This controller exposes the `Triad` as well as the `AnalyticsGoal` management methods through a RESTful API for HTTP.

Package: service

- **AnalyticsModulesService:** The service enables the Analytics Method macro component to manage the `Triad` objects as well as the `AnalyticsGoal` objects. It exposes its functionalities through the `AnalyticsMethodsController`. When saving `Triad` entities, it verifies that they do not already exist. The `Triad` objects do not vaidate the saved configurations. This means that validation must be performed before storing the `Triad`.

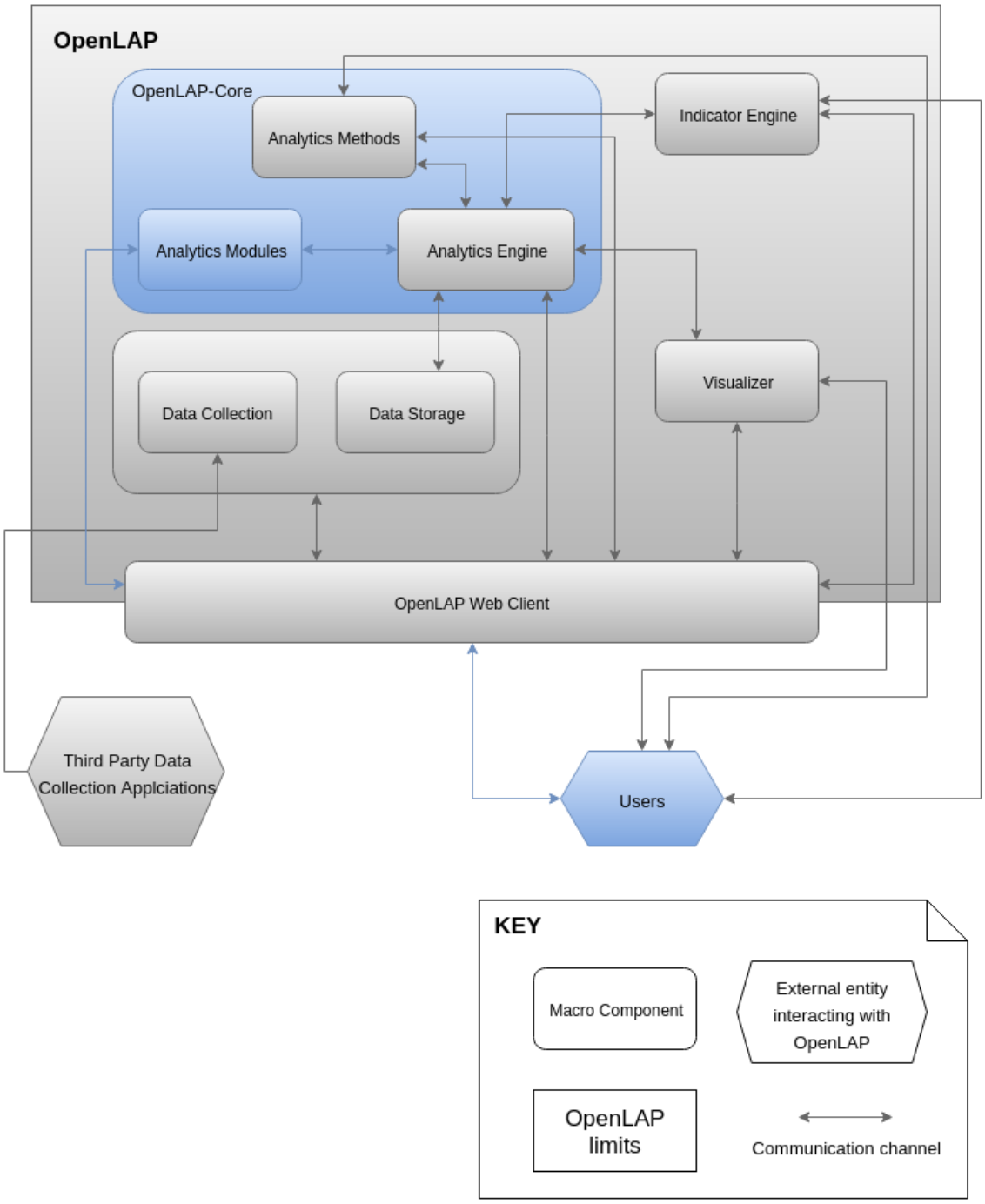
Package: dataAccess

- **TriadRepository:** A data access repository to handle database interaction and load the model of the `Triad` objects and it's related entities.
- **AnalyticsGoalRepository:** A data access repository to handle the database interaction in order to manage the Analytics Goal entities and its related Analytics Methods.

Package: model

- **Triad:** Triads represent groupings of Indicator, Analytics Methods and Visualizations. In this case, it only requires the references to the IDs of those elements. Additionally, the Triads store the `OLAPPortConfigurations` between both the Indicator output to the Analytics Method input and the Analytics Method output to Visualization Input.
- **IndicatorReferene:** The Indicator (specifically its ID in the `IndicatorEngine`) associated with the particular `Triad`.
- **AnalyticsMethodMetadata::** The AnalyticsMethodMetadata (specifically its ID in the `AnalyticsMethods` macro component) associated with the particular `Triad`.
- **VisualizationReference:** This model object holds the data for referencing a Visualization from the Visualizer macro-component.
- **AnalyticsGoal:** The Analytics Goal represents an arbitrary defined Analytics Goal that is created by the Users of type Teacher. The Analytics Goal correlates multiple Analytics Methods to itself and is used to make suggestions to the user when creating the Triad.

Context Diagram



Variability Guide

- Any General User can submit new Analytics Goals requests through the API. In order for Analytics Methods to be correlated with Analytics Goals, a user of type Administrator must activate (or authorize) the requested Analytics Goal. Once the Analytics Goal is active, it can be correlated with Analytics Methods. The `OpenLAPWebClient` should provide views for the submission of the new Analytics Goals as well as for activating them and correlating with existing Analytics Methods.
- The saving of new `Triad` objects are done through the Analytics Engine macro component. Ideally it is done only through that component since the Analytics Methods macro component does not verify the correctness of any configuration of the compatibility between the components of the `Triad`.

Rationale

The Analytics Methods macro component aims to be a simple referencing mechanism for making suggestions and gaining quick access for stored Triads. The Triads are stored as they are and the compatibility (the configuration of the triad) of the Indicator, Analytics Method and Visualization are not verified, since the task of doing so is done by the Analytics Engine macro component. The saving methods for storing `Triad` entities are designed to be used by the Analytics Engine alone. Methods for viewing the `Triad` objects can be made available to other requests sources that are not necessary the Analytics Engine.

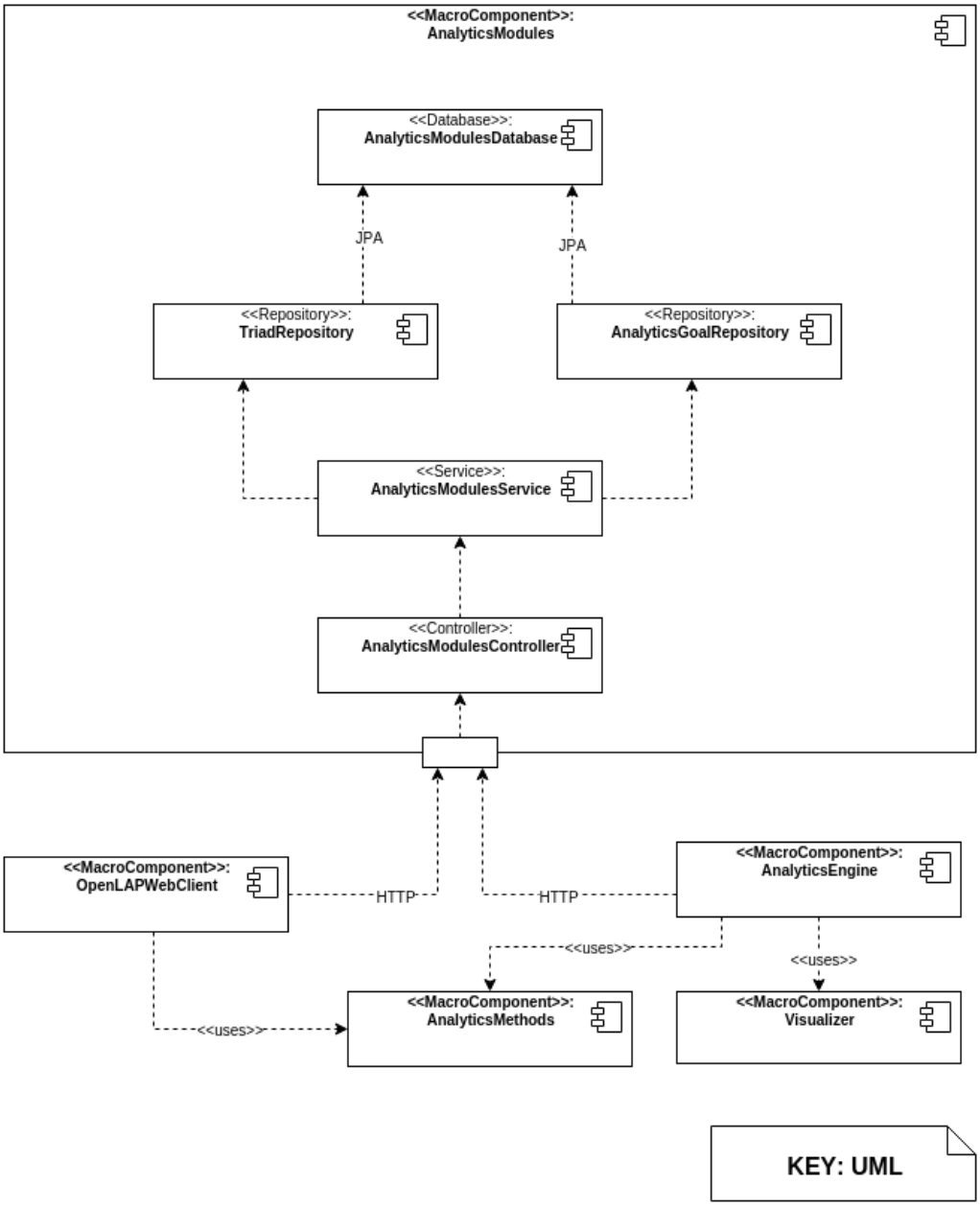
Additionally, the Analytics Methods allow the Users to create Analytics Goals. The Analytics Goals are entities that describe an arbitrary analysis goal such as personalization, comparison, etc. An Analytics Goal can aggregate Analytics Methods once it has been activated. The correlation between Analytics Goal and Analytics Methods are then saved and shown as suggestions during the creation of a `Triad`.

Related Views

- Section 4.6.2. describes the Component and Connector views for this macro module.
- Section 4.3. explains the Indicator Engine
- Section 4.5. explains the Analytics Methods
- Section 4.4. explains the Visualizer
- Section 4.2. explains the Analytics Engine

ANALYTICS MODULES: COMPONENT AND CONNECTOR VIEWS

Primary Presentation



Element Catalog

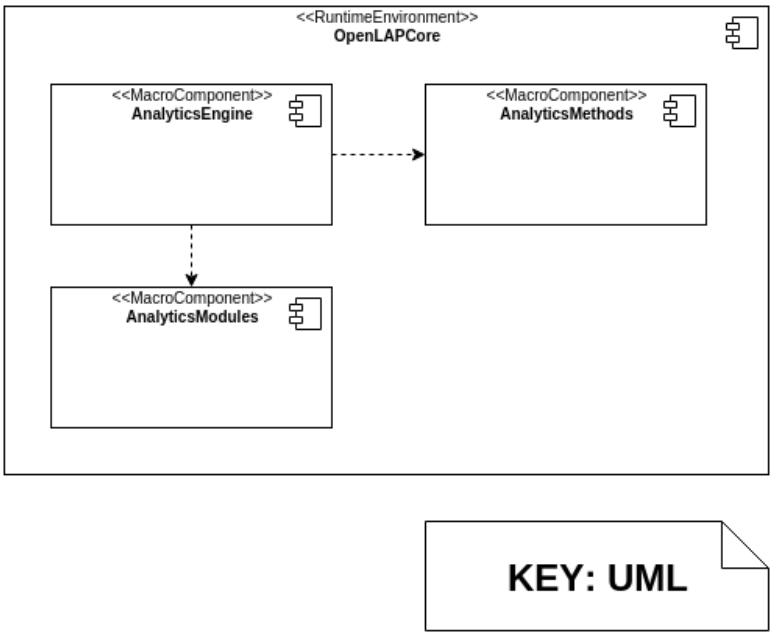
Components

- **<<Database>>AnalyticsModulesDatabase**: The database used to store the `Triad` and the `AnalyticsGoal` entities as well as the related objects.
- **TriadRepository**: The Repository will be accessed in order to provide a data access to the `Triad` objects in the Database. The database sill be accessed through JPA and will be local to the Analytics Modules macro component.
- **AnalyticsGoalRepository**: The Repository will be accessed in order to provide a data access to the `AnalyticsGoal` objects in the Database. The database sill be accessed through JPA and will be local to the Analytics Modules macro component.
- **AnalyticsModuleService**: The service executes direct calls over the repositories to save, update and retreive the different `Triad` and `AnalyticsGoal` objects as well as their related model entities.
- **AnalyticsModulesController**: The controller exposes the RESTful methods through HTTP. The Analytics Methods can be assigned to existing Analytics Goals, which contacts the service, retreives the `AnalyticsGoal` object and assigns it the reference to the `AnalyticsMethodMetadata`. In a similar manner, whenever a `Triad` save request is made, the controller assembles the `Triad` with the reference `AnalyticsMethodMetadata`, the `IndicatorReference`, and the `VisualizerReference`. Then handles it to the Service, which in turn will contact the repository to handle the storage of it into the database.
- **<<MacroComponent>>OpenLAPWebClient**: The macro component explained in section 4.7. It uses the RESTful methods exposed by the `AnalyticsModulesController` to access the methods of the macro component exposed here. It retrieves `Triad` entities as well as manages `AnalyticsGoals`. It also access the `<<MacroComponent>>AnalyticsMethods` to retreive the `AnalyticsMethodsMetadata` and associate it with selected `AnalyticsGoal` entities by the User of type Learner.
- **<<MacroComponent>>AnalyticsEngine**: The macro component explained in secion 4.2. It uses the It uses the RESTful methods exposed by the `AnalyticsModulesController` to access the methods of the macro component exposed here. It submits and retrieves Triads.
- **<<MacroComponent>>Visualizer**: The Visualizer macro component is used by the `<<MacroComponent>>AnalyticsEngine` to provide the references to the different visualizations.
- **<<MacroComponent>>AnalyticsMethods**: Similarly, the `<<MacroComponent>>AnalyticsMethods` is used by the `<<MacroComponent>>AnalyticsEngine` to provide the `AnalyticsMethodsMetadata` references of when creating `Triad` entities. Additionally, is accessed by the `<<MacroComponent>>OpenLAPWebClient` to associate the `AnalyticsMethodsMetadata` to existing `LarningGoal` entities.

Connectors

- **JPA**: The database access calls are made using JPA to the Database. A ORM can be used for it.
- **HTTP**: The calls to the controller are made by the HTTP RESTful methods exposed by it. The OpenLAPWebClient will offer an interface for the user to interact with the Analytics Modules macro component.
- **uses**: The Analytics Modules macro component will not have direct access over the instances of either the Analytics Modules or the Visualizers. The Analytics Engine will therefore provide information about their identifiers to the Analytics Modules and the Analytics Modules will store them in `Triad` objects. The objects stored are mere POJO representations for reference only.
- Other calls are made using traditional Java object calls.

Context Diagram



Variability Guide

- While the `AnalyticsGoals` can be added by Teacher Users, the Triads can only be created indirectly, when the `<<MacroComponent>>AnalyticsEngine` sends a request to save the references. The `<<MacroComponent>>AnalyticsEngine` also retreives the Triads whenever the user request to use an existing triad.

Rationale

The Analytics Methods macro component is a referencing component. It allows to perform two basic operations:

1. Manage Analytics Goals.
2. Manage Triads.

The way the different operations are approached are however binded to the functioning of the flow of the Analytics Engine. This is the reason behind some decisions such as:

- Storing references to the Analytics Method, Indicator and Visualization in a `Triad`. Since the only make sense in the exercise of executing the complete flow, the Triads only make sense in the context of the Analytics Engine macro component. The Analytics Engine macro component then is able to access the required Indicator, Analytics Methods and Visualizations through the APIs those macro components provide (RESTful or otherwise). This is also the reason behind the Analytics Modules macro component residing in the same run-time environment as the Analytics Engine, since the first one acts very much like a lookup table for the second.
- Allowing the access to the saving of `Triad` only to the AnayticsEngine. The creation of the `AnalyticsGoal` entities is however a manual task and is totally dependent on the Users (of type Teacher). This particular method, and the binding of the

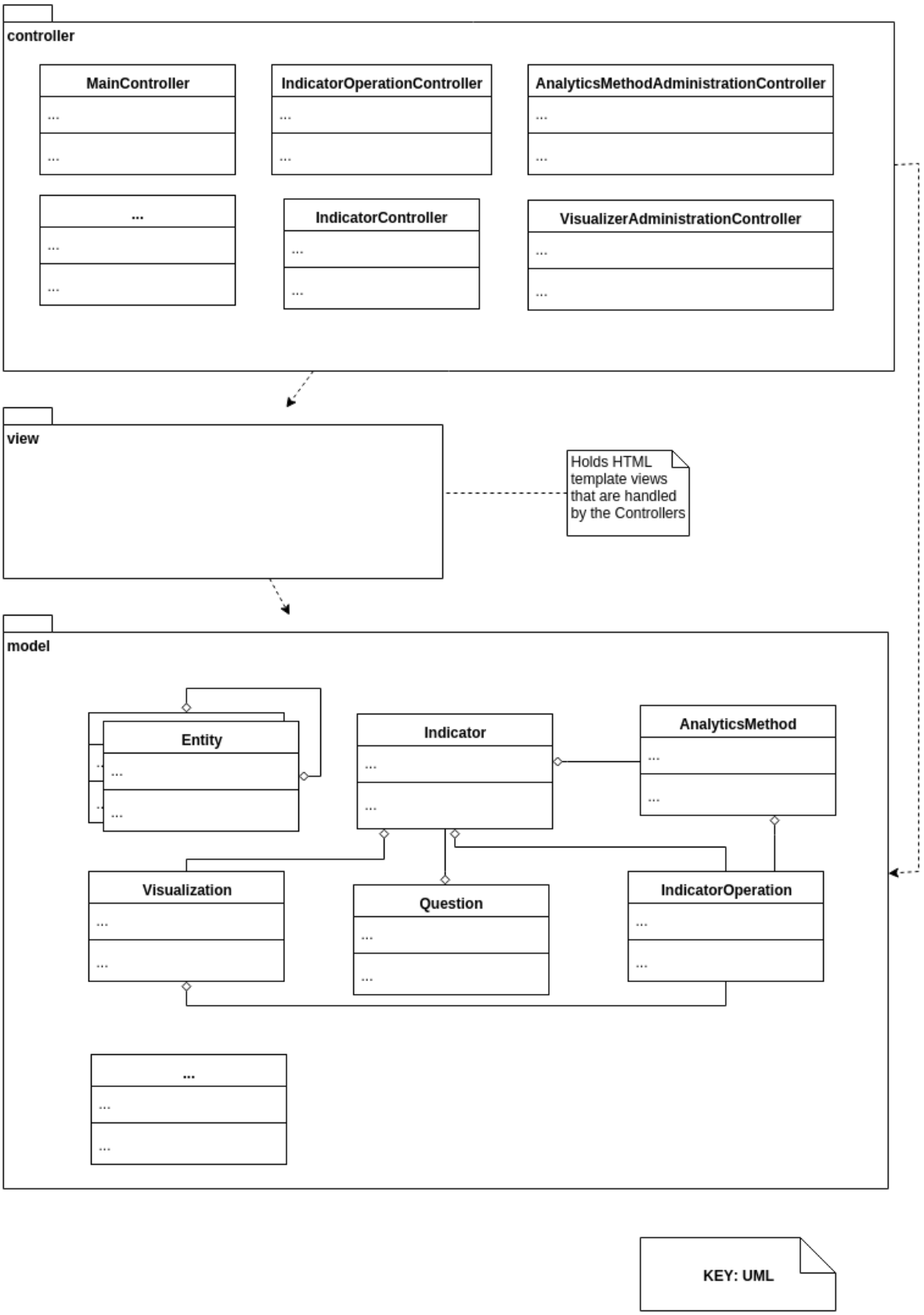
`AnalyticsMethodMetadata` references to the Analytics Goals are then accessible to the `OpenLAPWebClient`.

Related Views

- Section 4.6.1. describes the Module views for this macro module.
- Section 4.3. explains the Indicator Engine
- Section 4.5. explains the Analytics Methods
- Section 4.4. explains the Visualizer
- Section 4.2. Analytics Engine

WEB CLIENT: MODULE VIEWS

Primary Presentation



Element Catalog

Package: controller

This main package will hold the interaction points with the REST APIs of the OpenLAP system. The main entry points should communicate with the Indicator Engine, the Analytics Methods and the Visualizer in order to obtain the elements to manage Indicators, Analytic Modules and Visualizations as well as using the Visualizations and Analytics Methods when instantiating Indicators. The entry points are described on each of the macro components views in this architecture document.

- **MainController**: This controller handle the basic and main operations of the client.
- **IndicatorController**: This controller handles operations with the Indicators, present on the Indicator Engine macro component trough it's own controller endpoints. The Client interfaces directly with the endpoints of the Indicator Engine that handle the management (creation, viewing, modifying, updating, instantiating, etc.) of the Indicators in the Indicator Engine. This controller then relies the information with the models on the `model` package and presents it on the HTML pages that are contained in the `view` package of this macro component. Modifications on the model referring to Indicator Operations that need to be saved are pushed back to the Indicator Engine (the back-end) through this controller, which again access the Indicator Engine through the endpoints defined by the Indicator Engine macro component controllers.
- **IndicatorOperationController**: This controller handles operations with the Indicator Operations, present on the Indicator Engine macro component trough it's own controller endpoints. The Client interfaces directly with the endpoints of the Indicator Engine that handle the management (creation, viewing, modifying, updating, deleting, etc.) of the Indicator Operations during the creation of the Indicators in the Indicator Engine. This controller then relies the information with the models on the `model` package and presents it on the HTML pages that are contained in the `view` package of this macro component. Modifications on the model referring to Indicator Operations that need to be saved are pushed back to the Indicator Engine (the back-end) through this controller, which again access the Indicator Engine through the endpoints defined by the Indicator Engine macro component controllers.
- **AnalyticsMethodAdministrationController**: Similar to the previous controllers, this controller is responsible on mediating with the Analytics Methods macro component, which acts as a back-end. Whenever Analytics Methods are involved in the instantiation of indicators, creating or executing Triads, modifying or deleting Analytics Methods (as well as creating security operations over them) this controller mediates with the respective macro component. The `model` package will hold any data required to be shown in the respective `view` and update it to the back-end when necessary.
- **VisualizerAdministrationController**: This controller allows the client to communicate with the endpoints established on the Visualizer macro component. Whenever visualizations are involved in the process of creating or executing Triads, or when uploading new visualizations and/or performing modification operations or deletion of the visualizations, this controller will permit the user to use the client to do so by mediating with the respective endpoints of the Visualizer macro component.
- **Other Controllers**: Whenever the user requires operations that interact with other parts of the OpenLAP system, such as login, security, personalization, etc. then a controller must be established in the client to handle them. The additional controllers will also ideally communicate to the other OpenLAP macro components through endpoints to be consistent on the overall RESTful API oriented architecture of the system.

Package: view

This package does not hold classes but HTML views or templates that are to be used by the controller. The models in the mode package represent the data that the view holds.

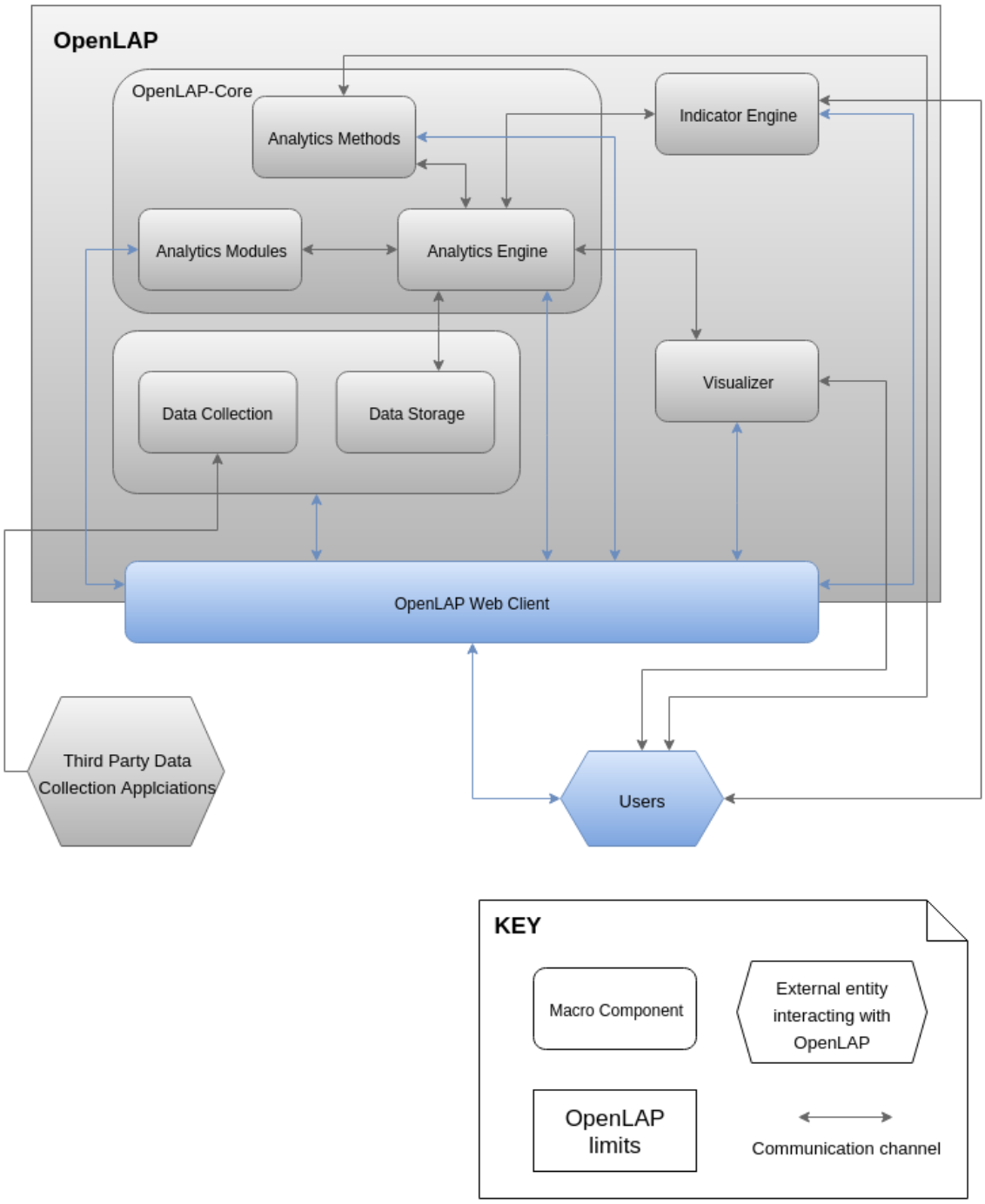
Package: model

This package holds the models required for the controller to operate. Note the models here are not necessary the same models as the other macro components. The model here are used to hold all the data required for the client to properly show the information required by and from the user. This can mean that the models can hold less or more data depending on the concrete implementation

of the controllers and views and could contain more models, split information of the other macro components (the back-end) into different models, extend it for interaction purposes or simply not use all of it.

- **Indicator**: This model holds the data required by the controller to mediate with the back-end macro component and to be displayed in the views related to Indicators (creation and instantiation of Indicators).
- **IndicatorOperation**: This model holds the data required by the controller to mediate with the back-end macro component and to be displayed in the views related to Indicator Operations (creation and instantiation of Indicators).
- **Question**: This model holds the data required by the controller to mediate with the back-end macro component and to be displayed in the views related to Questions (creation and instantiation of Indicators).
- **AnalyticsMethod**: This model holds the data required by the controller to mediate with the back-end macro component and to be displayed in the views related to Analytics Methods (instantiation of Indicators, creation and modification of Analytics Methods).
- **Visualization**: This model holds the data required by the controller to mediate with the back-end macro component and to be displayed in the views related to Visualizations (instantiation of Indicators, creation and modification of Visualizations).
- **Learning Context Data Model models**: Whenever Learning Context Data Model entities are involved, there must exist a model to hold the pertinent data for those operations.
- **Other models**: Additional models are required to give support operations, create logging, handle users, security, metrics and session related data.

Context Diagram



Variability Guide

- This macro component is designed not to represent exact connection points but to illustrate the purpose of the client. New controllers, models and views are to be added where needed.
- Operations on the Indicators will most likely be related to different views. Once different Indicator Operations are created, abstract controller must be created to handle the main purposes and overwrite or modify on child classes that handle the specifics of each IndicatorOperation controller.

Rationale

Using best practices and architectural standards is one of the principles of this architecture. For designing decoupled clients, one of the most effective approaches is to use the Model View Controller pattern. This pattern assigns a controller the responsibilities to communicate with server side components, fetch data, send queries and forward the results to the model and the views. The views where are HTML pages tied to the model, that hold the data fetched and transformed by the controller with which the user can interact with. Whenever the model is changed, the view reflects it and, when needed, the controller performs new server queries trough the API to push changes to the OpenLAP system. This allows not only to support a decoupled design from the main client to the server side components, but also enables an easier extension of the client and makes possible creating other clients, in other platforms with different functionalities.

Ideally, the client should be implemented using and MVC framework that, if possible, is compatible with the technologies used in through the project. This allows to easily maintain the client, be easily extended for new functionalities, and handle easily security, logging, deployment, testing and other non functional requirements. Most MVC frameworks offer powerful template systems to embed code into HTML views. The power of using HTML views is that allows the client to be updated easily through CSS and Javascript. This simplifies the design and implementation of the client an allows creating Usability changes without modifying the client logic.

The main idea is to leverage on the RESTful APIs offered by the macro components of the OpenLAP system. This way, the client can directly communicate with the different pieces of the system ad centralize the interaction of the users in a clean, easy to maintain and extend, way.

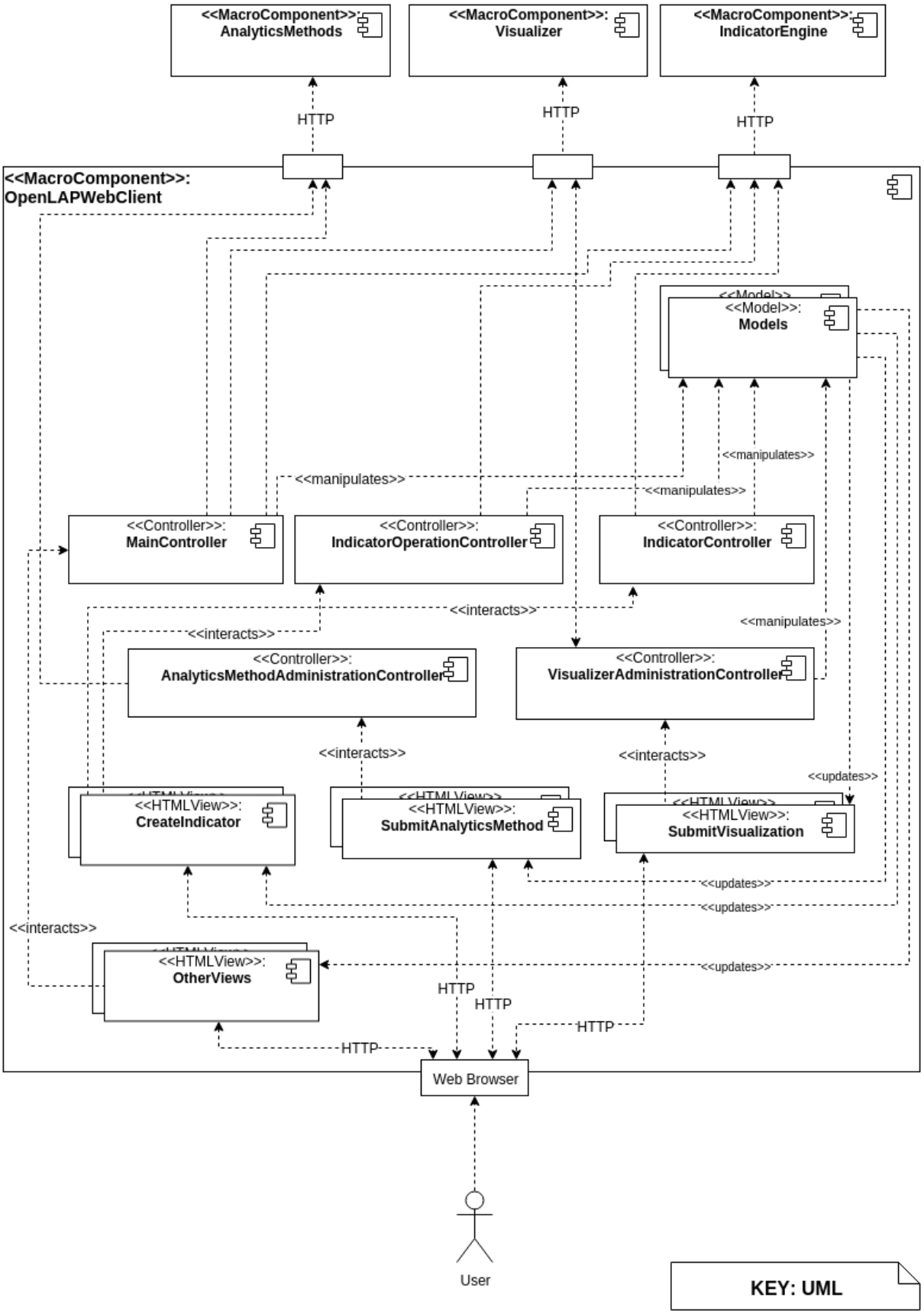
The list of components, modules and classes here show are just representational for the purposes of mapping the main elements to the MVC pattern and the implementation should be responsible of determining which exact realizations of the components are needed. New ones can be introduced, aiming for usability and covering the Use Cases.

Related Views

- Section 4.7.2.OpenLAP Web Client: C&C Views contains the Components and Connectors view description.
- Section 4.3 contains the Architecture of the Indicator Engine macro component and its RESTful API
- Section 4.4 contains the Architecture of the Visualizer macro component and its RESTful API
- Section 4.5 contains the Architecture of the Analytics Method macro component and its RESTful API

WEB CLIENT: COMPONENT AND CONNECTOR VIEWS

Primary Presentation



Element Catalog

Components

- **<<Controllers>>**: Controllers are intended to be a proxy on the MVC pattern. The Controller allows the user to access the back-end methods. The Controllers in the OpenLAP Web Client are divided on macro component they interact with and the specific functionalities they represent for the user. Each macro component that the user can interact with can be reached through the OpenLAP Web client and the proxy is a specific controller (more if needed, as for example the case with the Indicator Engine, which holds controllers for Indicators, Indicator Operations and Questions).
- **<<Views>>**: Views are HTML pages held on the OpenLAP Web Client. They will use a template system such as Thymeleaf, AngularJS, BackboneJS, JSPs or Freemaker. Ideally, the templates should allow binding to the model simplified implementation. The templates allow to use an extension over XML based languages to create requests to the model and embed the model into the HTML content with the help of the Controllers. Each view will represent at least a screen or interaction point between the user and the controller (and the model) and is rendered through a web browser using HTML.
- **<<Models>>**: Models allow the controller to access data locally by loading it through the macro components REST APIs by the controllers.
- **User**: Students, Teachers, Developers and Researchers are the main users of the OpenLAP Web Client. They Interact with the macro component through a Web browser. Not all of the options and views are available for all the users. For example, submitting Visualizations and Analytics Methods are only possible to Researchers and Developers
- **<<MacroComponents>>**: The back-end of the OpenLAP Web Client is the different macro components described in this architecture. Mainly and directly is the Indicator Engine (for creating, modifying, instantiating indicators), the Analytics Methods (submitting and managing Analytics Methods) and the Visualizer (Submitting and managing visualizations). Other macro components that expose API methods to the OpenLAP Web Client are able to be accessed through REST in the form of HTTP endpoints.

Connections

- **HTTP**: HTTP is used to communicate both with the web browser from the macro component controller and views to the web browser (users) and between the OpenLAP Web Client and other macro components (through their RESTful API endpoints). In particular for the Users, the HTTP is used to modify and interact with the Views. The views are templates that interpolate and bind with the models, whenever the User interacts with the Views, the Controllers takes an action and modifies the Models accordingly.
- **<<interacts>>**: Whenever the User creates uses the HTTP views, the views interact with the Controllers, in particular, the controllers activate actions that modify the model directly or that trigger calls to the other macro components. Those call modify the Models, which in turn updates the Views for the User to interact.
- **<<manipulates>>**: The controllers manipulate the information stored on each of the models.
- **<<updates>>**: The models update the views with the information that the controller has modified.

Variability Guide

- The framework for implementing the MVC client is up to the implementation team to decide. While some frameworks rely all the execution to the client side, others can work between a server side code and render only the views at the client. MVC is a widespread principle and the details on which specific technologies to use to implement are not relevant to the architecture.
- Controllers can be redistributed with other responsibilities, however to support and reflect the architecture the way it was meant to be and to provide good modularity and maintenance, it is recommended that controllers reflect at least a subset of specific macro components. More than one controller can exist by macro component, but mixing different macro components into a single controller should be, as a general rule, avoided.

Rationale

The choice of using MVC does not only support an extensible and modular architecture for the client of the OpenLAP but also leverages directly on the RESTful modular approach of the other macro components involved, since it allows distribution of the components in different locations with minimal impact. Also permits other possible clients to be designed and implemented, since the endpoints of the macro components are accessible to authorized users and the OpenLAP Web Client is, in this regard, a convenience but not the only option to utilize the OpenLAP environment (or parts of it).

For the Views, most common MVC frameworks integrate one or many template engines. Using such a template system like Thymeleaf, JSP, AngularJS, BackboneJS or similar allows to use an extension of HTML to provide the views to the users. This enables the benefits of HTML and web technologies to support the rendering of information to the users web browser.

Related Views

- Section 4.7.1.contains the Package and Module view description in a static manner.
- Section 4.3 contains the Architecture of the Indicator Engine macro component and its RESTful API
- Section 4.4 contains the Architecture of the Visualizer macro component and its RESTful API
- Section 4.5 contains the Architecture of the Analytics Method macro component and its RESTful API

RATIONALE

The purpose of this architecture is to convey a high level of understanding of the internals of the OpenLAP, explain its main components, and outline their main responsibilities and interactions in order to support the realization of the ecosystem proposed in the paper that envisions this platform. With these architecture, the developers should be able to find the general purpose and architectural approach guidelines when implementing the macro components of the OpenLAP.

The purpose of the architecture is NOT:

- To provide a high level of detail of each of the components of the OpenLAP.
- To be a formalization of the components, classes, and methods of each of the Macro Components of the OpenLAP.
- To establish or impose specific technologies for implementation.
- To be followed literally, since many details are left unspecified on purpose.

Developed are expected to understand this high level architecture but also are responsible of implementing the components to the best of their judgment. The OpenLAP is composed of a set of intercommunicating components (referred as Macro Components) are based on the components explained in the vision of Toward an Open Learning Analytics Ecosystem: an Analytics Engine, a Visualizer, an Indicator Engine, A series of Analytics Modules, Analytics Methods and a Database. This is the basis of the division of the OpenLAP into Macro Components: it does not only to map the components described on the paper, but also provide an architectural decision by creating a modular system were components have specific responsibilities, are relatively independent of each other and in conjunction flesh out the purpose if the OpenLAP. This supports the principle of high cohesion and low coupling that most large scale systems should strive towards. Each of the macro components does not only provide an abbreviation of a functionality of the system, but is also a sub-unit of architectural value in the sense that it is internally designed taking into account its responsibilities of the whole system but at the same time following similar patterns of design.

A popular approach on the current web that has shown to be effective, horizontally scalable and supported by multiple technologies is the 3-layer (or n-layer) architectural approach. By providing each component with a somewhat self sufficient and robust but simple architecture of businesses logic, data access and facades it is possible to interconnect relatively simple components that would otherwise be too complex if they were all deployed in a single unit. Additionally other enables that the components can be individually developed and interconnected loosely. In order to achieve this interconnection, each of the macro component with a server side exposes an HTTP REST API to either users or other components.

MAPPING REQUIREMENTS TO ARCHITECTURE

The mapping to the functional requirements that is exposed in the System Overview can be seen in the table below.

Requirement	Corresponding Architecture Views and Explanation
OpenLAP_SUC_01: View/Modify Dashboard	• Views on Section 4.1. Data Collection and Data Storage: The Data Collection and Data Storage Macro Module is involved in handling the Query data when the Analytics Engine executes the Query that the Indicator of the Triad provides. • Views on Section 4.2. Analytics Engine: The Analytics Engine is in charge of processing the Triads that are to be processed and subsequently shown in the Dashboard (or any other client). • Views on Section 4.3. Indicator Engine: The main entry point to the Indicator Engine form the Client is the Dashboard. The Indicator Engine is the first step to create the Elements shown in the dashboard if there are any indicators. • Views on Section 4.4. Visualizer: When the user selects to execute a Triad, the Visualization involved will be fired up by the Analytics Engine and process the data sent by the Indicator's query or the Analytics Method. The result will be forwarded to the Analytics Engine, which will return it to the requester. • Views on Section 4.5. Analytics Modules: When the user selects to execute a Triad, the Analytics Method involved will be fired up by the Analytics Engine and process the data sent by the Indicator's query. • Views on Section 4.6. Analytics Modules: When the user is creating the new Indicator and is presented the option for creating a triad along with an Analytics Method and a Visualization, the Analytics Modules suggest the Analytics Methods based on the Analytics Goal. Additionally the Analytics Methods will store the Triad for future reference. • Views on Section 4.7. OpenLAP Web Client: The client provides a view to see the dashboard. The client also is able to make the interactions possible to show the results of the Analytics Engine.
OpenLAP_SUC_02: Create Question	• Views on Section 4.3. Indicator Engine: The Questions are saved into the Indicator Engine and related to indicators accordingly. • Views on Section 4.6. Analytics Modules: The Analytics Modules correlates the Triads to answer the questions. • Views on Section 4.7. OpenLAP Web Client: The client will provides the GUI in where questions are asked and then associated with indicators.
OpenLAP_SUC_03: Associate Indicator to Question	• Views on Section 4.3. Indicator Engine: The Indicators are either created to or retreived from the Indicator Engeine. New Indicators are saved along with their filters. Indicators used as templates are initially fetched to be used and saved as new Indicators. • Views on Section 4.7. OpenLAP Web Client: The client will provide a web GUI to configure and save Indicator along its filters.
OpenLAP_SUC_04: Generate Indicator	• Views on Section 4.3. Indicator Engine: The Indicator Engine provides the facilities to create the query and filters for which the Indicator will be created and saved. • Views on Section 4.7. OpenLAP Web Client: The client will provide a web GUI to configure and save Indicator along its filters.
OpenLAP_SUC_05: Generate Triad	• Views on Section 4.2. Analytics Engine: The analytics Engine will orchestrate the whole procedure of creating the triad as explained in the views. In particular, the C&C view illustrates the components it communicates with in order to create and save the Triad. • Views on Section 4.3. Indicator Engine: The Indicator Engine provides the Indicator to be referenced in the Triad.

	• Views on Section 4.4. Visualizer: The Visualizer provides the Visualization that is to be referenced in the Triad. It also validates that the configuration of output of the Analytics Method is compatible with the Visualization input. • Views on Section 4.5. Analytics Methods: The Analytics Methods will provide the particular Analytics Method that is to be referenced by theTriad. It also validates that the configuration of output of the Indicator is compatible with the Analytics Method input. • Views on Section 4.6. Analytics Modules: The Analytics Modules is where the Triad will be saved. • Views on Section 4.7. OpenLAP Web Client: The client will provide a web GUI to configure and save the Triad.
OpenLAP_SUC_06: View Triad Execution	• Views in 4.1. Data Collection and Data Storage: The data of the query stored in the Indicator is held in this component. The Analytics Engine executes the query and handles the result to the appropriate macro components. • Views in 4.2. Analytics Engine: Accepts the request to execute Triads, which contains reference to the Indicator, Analytics Method and Visualization to be used. It then orchestrates the handling of the workflow, where the query is executed, its results sent to the Analytics Method and executed and then finally sent to the Visualizer to process. It then returns the generated HTML code. • Views in 4.3. Indicator Engine: The Indicator Engine has an existing Indicator that exists on a Triad. The indicator has a query to fetch the data which is executed by the Analytics Engine. • Views in 4.4. Visualizer: The Visualizer uses the visualization stated on the Triad. The macro component receives the data that is outputed from the Analytics Method or Indicator's query and handled by the Analytics Engine. It processes the data and returns the visualization code that can be embedded to the Analytics Engine. • Views in 4.5. Analytics Methods: The Data that is outputed by the execution of the Indicator query is passed by the Analytics Engine to an Analytics Method for processing. The result is handled back to the Analytics Engine to be forwarded to the Visualizer. • Views in 4.6. Analytics Modules: The Analytics Module holds the Triad that is to be executed to the Analytics Engine. It is this Triad that indicates the Analytics Engine which Indicator, Analytics Method and Visualization to use at execution.
OpenLAP_SUC_07: Administer Data Collection and Data Availability	• Views on Section 4.1. Data Collection and Data Storage: The Data Collection and Data Storage macro component provides the main functionalities to realize this use case. It enables the users to modify the data collection settings and to configure existing collectors to work on their accounts. • Views on Section 4.7.OpenLAP Web Client: The client should provide a GUI for the users to modify existing data collectors attached to their accounts as well as attaching existing data collectors to their accounts.
OpenLAP_SUC_08: Request Creation/View Analytics Goals	• Views in 4.6. Analytics Modules: The Analytics Modules exposes an API that allows the Users to request the creation of new Analytics Goals as well as attaching Analytics Methods to them. Additionally it provides means to activate Analytics Goals for Administrator users. • Views in 4.7. MVC Client: The client has a GUI to access the HTTP RESTful methods that the Analytics Modules expose through their API.
OpenLAP_SUC_09: Register Data collection for Third Party Application	• Views on Section 4.1. Data Collection and Data Storage: The Data Collection and Data Storage macro component provides the methods to submit the authorization of a data collection third party program. In particular, holds HTTP RESTful methods to register and authorize the application through the controller. • Views on Section 4.7. OpenLAP Web Client: The client provides a GUI for the users of type Developer/Researcher to register their external applications for collecting Data. The GUI, as with other components and Use Cases, is a facade to the HTTP RESTful methods exposed by the other macro components.
OpenLAP_SUC_10: Create/View/Update Analytics Methods	• Views in 4.5 Analytics Methods: The Analytics Methods macro component enables the Users to submit Analytics Methods through the Controller. Additionally enables the user of type Developer/Researcher to obtain the authorization key to sign the JAR files submitted. • Views in 4.7. OpenLAP Web Client: The web client has an interface that connects to the Analytics Method

	Controller handles the upload requests, the application key requests and the Analytics Method modification requests.
OpenLAP_SUC_11: Create/View/Update Visualizations	• Views in 4.4 Visualizer: The Visualizer macro component enables the Users to submit Visualizations through the Controller, which exposes an API. • Views in 4.7. OpenLAP Web Client: The client has an interface that sends RESTful requests to the API of the Visualizer and handles the upload requests as well as the modification requests.
OpenLAP_SUC_12: Submit Usage Data	• Views in 4.1. Data Collection and Data Storage: This macro component offers an API accessible through HTTP so external applications can submit usage data. The applications that are registered can use the API in order to submit data, which is validated by the Data Collection and Data Storage macro component and stored whenever there are no errors present.

OPEN QUESTIONS

- **Deployment views:** The deployment views were not included in the different modules since they are specific to the implementation details. The concrete implementation is not provided in this high level architecture. However some suggestions are made in order to preserve the main functioning of the system as a complete entity. In particular, the deployment of the Analytics Engine, Analytics Methods and Analytics Modules in the same run-time environment is vital to the architecture.
- **Security Mechanisms:** The platform does not specify the security mechanisms of the platform and are left for the implementation. This is typically handled by the frameworks used for implementation. The security mechanism should be treated twofold:

    - **User Access Control:** Users should be able to log into the system and use its capabilities through an account.
    - **API Key:** Developers and Researchers should be able to obtain an API key to use and execute the different methods provided by the complete API of the OpenLAP.
- **Concatenation of Analytics Methods:** At the point this documentation is produced, the concatenation of different Analytics Methods although theoretically possible, it is not contemplated by the workflow of the Analytics Engine. The Analytics Engine could, in later iterations, provide mechanisms to extend Triads and allow to use multiple Analytics Methods over a single execution.

GLOSSARY

- **Analytics Goal**: An Analytics Goal represents a particular set of goals for a Teacher or for a Researcher/Developer. It is represented in the OpenLAP system as a description of the Analytics Goal itself and a correlation of Analytics Method. Whenever the Analytics Goal is selected for a particular interaction, the associated Analytics Methods will be shown as suggestions for creating the Triad.
- **Configuration**: A configuration represents the mapping between the outputs and inputs of the different components of a Triad. Normally refers to the configuration of all the components in a particular Triad, i.e. the mapping between the output of an Indicator to the Input of an Analytics Method as well as the output of the same Analytics Method to the input of the Visualization specified in the Triad. A configuration has its own model and methods to guarantee that the data types are compatible. The configuration can be done manually, or if the types are compatible, left as default. The configuration influences how the data is interpreted in both the Analytics Method and the Visualization.
- **Macro Component**: This refers to one of the seven macro components of the Open Learning Analytics Platform: Data Collection and Data Storage, Analytics Engine, Indicator Engine, Visualizer, Analytics Methods, Analytics Modules, OpenLAP Web Client.
- **Question**: A Question represents something the user of the OpenLAP wants to answer using the available data. The question will be written in natural form and the OpenLAP will attempt to answer the question either by using existing Indicators or by creating a new one with the help of the user. Questions are available to other users after they are created.
- **Triad**: A triad consist of a grouping of the references to an Indicator, an Analytics Method and a Visualization.

Acronyms

- **C&C**: Stands for "Components and Connectors". The view that describes the components and connectors of the different Macro Components.

Naming Conventions

Images

- Architecture figures: `OLAP_[Section]_[name].[format]`

Use Cases

- Main Scenario Use Cases: `OLAP_SUC_[Number]`, where `SUC` stands for Scenario Use Case (for the main scenarios, other additional Use Cases developed should use `UC` instead.

REFERENCED MATERIAL ON .BIB FORMAT

```
@mastersthesis{tanmayaThesis,
author = {Mahapatra, Tanmaya},
title = {A Rule-Based Indicator Definition Tool For Personalized Learning
  Analytics},
school = {RWTH, Aachen},
year = {2015}
}


@mastersthesis{bassimThesis,
author = {Bashir, Bassim},
title = {Modular Visualization Framework for the Open Learning Analytics
   Platform},
school = {RWTH, Aachen},
year = {2015}
}

@mastersthesis{memoonaThesis,
author = {Mughal, Memoona},
title = {Design and Implementation of an Indicator Editor for Personalized
  Learning Analytics},
school = {RWTH, Aachen},
year = {2015}
}

@mastersthesis{mareikeThesis,
author = {Bültmann, Mareike},
title = {Design Und Usability-Studie Eines Learning Analytics Werkzeugs},
school = {RWTH, Aachen},
year = {2011}
}

@article{chatti2015learningecosystem,
title={Toward an Open Learning Analytics Ecosystem},
author={Chatti and Muslim, Schroeder},
journal={Springer},
year={2015},
volume={ }
}

@article{chatti2012reference,
title={A reference model for learning analytics},
author={Chatti, Mohamed Amine and Dyckhoff, Anna Lea and Schroeder, Ulrik and
```

```
  Th{\"u}s, Hendrik},
journal={International Journal of Technology Enhanced Learning},
volume={4},
number={5},
pages={318--331},
year={2012},
publisher={Inderscience}
}

@article{chatti2014learning,
title={Learning Analytics: Challenges and Future Research Directions},
author={Chatti, Mohamed Amine and Lukarov, Vlatko and Th{\"u}s, Hendrik and
Muslim, Arham and Yousef, Ahmed Mohamed Fahmy and Wahid, Usman and Greven,
Christoph and Chakrabarti, Arnab and Schroeder, Ulrik},
year={2014}
}

@article{ferguson2012learning,
title={Learning analytics: drivers, developments and challenges},
author={Ferguson, Rebecca},
journal={International Journal of Technology Enhanced Learning},
volume={4},
number={5-6},
pages={304--317},
year={2012},
publisher={Inderscience Publishers}
}

@article{norris2008action,
title={Action Analytics: Measuring and Improving Performance that Matters in
  Higher Education.},
author={Norris, Donald and Baer, Linda and Leonard, Joan and Pugliese, Louis
  and Lefrere, Paul},
journal={EDUCAUSE review},
volume={43},
number={1},
pages={42},
year={2008},
publisher={ERIC}
}

@article{goldstein2005academic,
title={Academic analytics: The uses of management information and technology
  in higher education},
author={Goldstein, Philip J and Katz, Richard N},
year={2005},
publisher={Educause}
}

@article{kruchten1995architectural,
title={Architectural Blueprints—The “4+ 1” View Model of Software Architecture},
author={Kruchten, Philippe},
```

```
journal={Tutorial Proceedings of Tri-Ada},
volume={95},
pages={540--555},
year={1995}
}

@article{laitinen1996estimating,
title={Estimating understandability of software documents},
author={Laitinen, Kari},
journal={ACM SIGSOFT Software Engineering Notes},
volume={21},
number={4},
pages={81--92},
year={1996},
publisher={ACM}
}

@techreport{kazman2000atam,
title={ATAM: Method for architecture evaluation},
author={Kazman, Rick and Klein, Mark and Clements, Paul},
year={2000},
institution={DTIC Document}
}

@article{guazzelli2010pmml,
title={What is PMML? Explore the power of predictive analytics and open
   standards},
author={Guazzelli, A},
journal={IBM developerWorks},
year={2010}
}

@article{guazzelli2009pmml,
title={PMML: An open standard for sharing models},
author={Guazzelli, Alex and Zeller, Michael and Lin, Wen-Ching and Williams,
  Graham},
journal={The R Journal},
volume={1},
number={1},
pages={60--65},
year={2009}
}

@phdthesis{siemens2011open,
title={Open Learning Analytics: an integrated \& modularized platform},
author={Siemens, George and Gasevic, Dragan and Haythornthwaite, Caroline and
  Dawson, Shane and Shum, Simon Buckingham and Ferguson, Rebecca and Duval,
  Erik and Verbert, Katrien and Baker, RSJD},
year={2011},
school={Open University Press}
}
```

```
@misc{apereoLAProcessorWiki,
title = {Apereo Learning Analytics Processor Wiki},
month = August,
year = {2015},
howpublished={\url{https://confluence.sakaiproject.org/display/LAI/Apereo+
  Learning+Analytics+Processor}}
}

@article{meier2009microsoft,
title={Microsoft application architecture guide},
author={Meier, JD and Hill, David and Homer, A and Jason, Taylor and Bansode, P
   and Wall, L and Boucher Jr, R and Bogawat, A},
year={2009},
publisher={Microsoft Press Book}
}

@misc{apereoHomePageIncubatorLAI,
title = {Apereo Home Page Incubator Project on Learning Analytics Initiative},
month = August,
year = {2015},
howpublished={\url{https://www.apereo.org/communities/learning-analytics-
  initiative}}
}

@article{mccabe1976complexity,
title={A complexity measure},
author={McCabe, Thomas J},
journal={Software Engineering, IEEE Transactions on},
number={4},
pages={308--320},
year={1976},
publisher={IEEE}
}

@article{mccabe1989design,
title={Design complexity measurement and testing},
author={McCabe, Thomas J and Butler, Charles W},
journal={Communications of the ACM},
volume={32},
number={12},
pages={1415--1425},
year={1989},
publisher={ACM}
}

@article{henderson1996coupling,
title={Coupling and cohesion (towards a valid metrics suite for object-oriented
  analysis and design)},
author={Henderson-Sellers, Brian and Constantine, Larry L and Graham, Ian M},
journal={Object Oriented Systems},
volume={3},
number={3},
```

pages={143--158},
year={1996}
}

@book{hitz1995measuring,
title={Measuring coupling and cohesion in object-oriented systems},
author={Hitz, Martin and Montazeri, Behzad},
year={1995},
publisher={Citeseer}
}

@article{jayaprakash2014early,
title={Early alert of academically at-risk students: An open source analytics
initiative},
author={Jayaprakash, Sandeep M and Moody, Erik W and Laur{\'\i}a, Eitel JM and
Regan, James R and Baron, Joshua D},
journal={Journal of Learning Analytics},
volume={1},
number={1},
pages={6--47},
year={2014}
}

@misc{ims2013learning,
title={Learning Measurement for Analytics Whitepaper},
author={IMS Global Learning Consortium and others},
year={2013}
}

@article{advanced2001sharable,
title={Sharable Content Object Reference Model (SCORM™)},
author={Advanced Distributed Learning Initiative and others},
journal={Advanced Distributed Learning},
year={2001}
}

@article{vassiliadis2009survey,
title={A survey of Extract--transform--Load technology},
author={Vassiliadis, Panos},
journal={International Journal of Data Warehousing and Mining (IJDWM)},
volume={5},
number={3},
pages={1--27},
year={2009},
publisher={IGI Global}
}

@article{romero2007educational,
title={Educational data mining: A survey from 1995 to 2005},
author={Romero, Cristobal and Ventura, Sebastian},
journal={Expert systems with applications},
volume={33},

```
number={1},
pages={135--146},
year={2007},
publisher={Elsevier}
}

@inproceedings{duval2011attention,
title={Attention please!: learning analytics for visualization and
  recommendation},
author={Duval, Erik},
booktitle={Proceedings of the 1st International Conference on Learning
  Analytics and Knowledge},
pages={9--17},
year={2011},
organization={ACM}
}

@book{croll2009complete,
title={Complete web monitoring: watching your visitors, performance,
  communities, and competitors},
author={Croll, Alistair and Power, Sean},
year={2009},
publisher={" O'Reilly Media, Inc."}
}

@article{barker2005ieee,
title={What is ieee learning object metadata/ims learning resource metadata},
author={Barker, Phil},
journal={cetis standards briefings series},
year={2005}
}

@inproceedings{lukarov2014data,
title={Data Models in Learning Analytics},
author={Lukarov, Vlatko and Chatti, Mohamed Amine and Th{\"u}s, Hendrik and
Kia, Fatemeh Salehian and Muslim, Arham and Greven, Christoph and Schroeder, Ulrik},
booktitle={DeLFI Workshops},
pages={88--95},
year={2014},
organization={Citeseer}
}

@article{cooper2014learning,
title={Learning Analytics Interoperability-The Big Picture In Brief},
author={Cooper, Adam},
journal={Learning Analytics Community Exchange},
year={2014}
}

@article{cooper2013learning,
  title={Learning Analytics Interoperability-a survey of current literature
    and candidate standards},
```

```
  author={Cooper, Adam R},
  year={2013}
}

@article{boehm1996identifying,
title={Identifying quality-requirement conflicts},
author={Boehm, Barry and In, Hoh},
journal={IEEE software},
volume={13},
number={2},
pages={25},
year={1996},
publisher={IEEE Computer Society}
}

@inproceedings{sefika1996monitoring,
  title={Monitoring compliance of a software system with its high-level design
    models},
  author={Sefika, Mohlalefi and Sane, Aamod and Campbell, Roy H},
  booktitle={Proceedings of the 18th international conference on Software
    engineering},
  pages={387--396},
  year={1996},
  organization={IEEE Computer Society}
}

@inproceedings{poulin1994measuring,
  title={Measuring software reusability},
  author={Poulin, Jeffrey S},
  booktitle={Software Reuse: Advances in Software Reusability, 1994.
    Proceedings., Third International Conference on},
  pages={126--138},
  year={1994},
  organization={IEEE}
}

@inproceedings{lenzerini2002data,
  title={Data integration: A theoretical perspective},
  author={Lenzerini, Maurizio},
  booktitle={Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART
    symposium on Principles of database systems},
  pages={233--246},
  year={2002},
  organization={ACM}
}

@article{brooke1996sus,
  title={SUS-A quick and dirty usability scale},
  author={Brooke, John and others},
  journal={Usability evaluation in industry},
  volume={189},
  number={194},
```

```
  pages={4--7},
  year={1996},
  publisher={London}
}

@article{fayad1997object,
  title={Object-oriented application frameworks},
  author={Fayad, Mohamed and Schmidt, Douglas C},
  journal={Communications of the ACM},
  volume={40},
  number={10},
  pages={32--38},
  year={1997},
  publisher={ACM}
}

@article{zhu2002information,
  title={Information transparency in electronic marketplaces: Why data
    transparency may hinder the adoption of B2B exchanges},
  author={Zhu, Kevin},
  journal={Electronic markets},
  volume={12},
  number={2},
  pages={92--99},
  year={2002},
  publisher={Taylor \& Francis}
}

JSONRFC
@article{crockford2006javascript,
  title={The JavaScript Object Notation (JSON) Data Interchange Format},
  author={Crockford, Douglas},
  journal={The Internet Engineering Task Force RFC},
  volume={7159},
  pages={1--15},
  year={2006}
}

@misc{fowler2004inversion,
  title={Inversion of control containers and the dependency injection pattern},
  author={Fowler, Martin},
  year={2004}
}

@inproceedings{boicea2012mongodb,
  title={MongoDB vs Oracle--Database Comparison},
  author={Boicea, Alexandru and Radulescu, Florin and Agapin, Laura Ioana},
  booktitle={2012 Third International Conference on Emerging Intelligent
    Data and Web Technologies},
  pages={330--335},
  year={2012},
  organization={IEEE}
```

```
}


@book{bachmann2011documenting,
title={Documenting Software Architectures: Views and Beyond},
author={Bachmann, Felix and Bass, Len and Garlan, David and Ivers, James
  and Little, Reed and Merson, Paulo and Nord, Robert and Stafford, Judith},
year={2011},
publisher={Addison-Wesley Professional}
}

@book{fowler2002patterns,
title={Patterns of enterprise application architecture},
author={Fowler, Martin},
year={2002},
publisher={Addison-Wesley Longman Publishing Co., Inc.}
}

@book{taylor2009software,
title={Software architecture: foundations, theory, and practice},
author={Taylor, Richard N and Medvidovic, Nenad and Dashofy, Eric M},
year={2009},
publisher={Wiley Publishing}
}

@book{gamma1994design,
title={Design patterns: elements of reusable object-oriented software},
author={Gamma, Erich and Helm, Richard and Johnson, Ralph and Vlissides, John},
year={1994},
publisher={Pearson Education}
}

@book{martin1983managing,
title={Managing the data-base environment},
author={Martin, James and Savant Institute},
year={1983},
publisher={Prentice-Hall Englewood Cliffs (NJ)}
}

@book{turnquist2014learning,
title={Learning Spring Boot},
author={Turnquist, Greg L},
year={2014},
publisher={Packt Publishing Ltd}
}

@book{martin2003agile,
title={Agile software development: principles, patterns, and practices},
author={Martin, Robert Cecil},
year={2003},
publisher={Prentice Hall PTR}
}
```

```
@article{mccabe1976complexity,
title={A complexity measure},
author={McCabe, Thomas J},
journal={Software Engineering, IEEE Transactions on},
number={4},
pages={308--320},
year={1976},
publisher={IEEE}
}

@book{masse2011rest,
title={REST API design rulebook},
author={Masse, Mark},
year={2011},
publisher={" O'Reilly Media, Inc."}
}

@article{johnson2004spring,
title={The Spring Framework--Reference Documentation},
author={Johnson, Rod and Hoeller, Juergen and Donald, Keith and Sampaleanu,
  Colin and Harrop, Rob and Risberg, Thomas and Arendsen, Alef and Davison,
  Darren and Kopylenko, Dmitriy and Pollack, Mark and others},
journal={Interface},
volume={21},
year={2004}
}

@book{meyer1988object,
  title={Object-oriented software construction},
  author={Meyer, Bertrand},
  volume={2},
  year={1988},
  publisher={Prentice hall New York}
}

@book{widenius2002mysql,
  title={MySQL reference manual: documentation from the source},
  author={Widenius, Michael and Axmark, David},
  year={2002},
  publisher={" O'Reilly Media, Inc."}
}

@article{bauer2005hibernate,
  title={Hibernate in action},
  author={Bauer, Christian and King, Gavin},
  year={2005},
  publisher={Manning Greenwich CT}
}

@book{lindholm2014java,
  title={The Java virtual machine specification},
```

```
  author={Lindholm, Tim and Yellin, Frank and Bracha, Gilad and Buckley, Alex},
  year={2014},
  publisher={Pearson Education}
}

@article{liang1998dynamic,
  title={Dynamic class loading in the Java virtual machine},
  author={Liang, Sheng and Bracha, Gilad},
  journal={Acm sigplan notices},
  volume={33},
  number={10},
  pages={36--44},
  year={1998},
  publisher={ACM}
}

@book{banker2011mongodb,
  title={MongoDB in action},
  author={Banker, Kyle},
  year={2011},
  publisher={Manning Publications Co.}
}

@book{anderson2010couchdb,
  title={CouchDB: the definitive guide},
  author={Anderson, J Chris and Lehnardt, Jan and Slater, Noah},
  year={2010},
  publisher={" O'Reilly Media, Inc."}
}

@book{muschko2014gradle,
  title={Gradle in Action},
  author={Muschko, Benjamin},
  year={2014},
  publisher={Manning}
}

@book{sobell2005practical,
  title={A practical guide to Linux commands, editors, and shell programming},
  author={Sobell, Mark G},
  year={2005},
  publisher={Prentice Hall Professional Technical Reference}
}

@book{blum2008linux,
  title={Linux command line and shell scripting bible},
  author={Blum, Richard},
  volume={481},
  year={2008},
  publisher={John Wiley \& Sons}
}
```

```
@misc{saunders2006intellij,
  title={IntelliJ IDEA in Action},
  author={Saunders, Stephen and Fields, DK and Belayev, E},
  year={2006},
  publisher={Manning}
}

@book{rumbaugh2004unified,
  title={Unified Modeling Language Reference Manual, The},
  author={Rumbaugh, James and Jacobson, Ivar and Booch, Grady},
  year={2004},
  publisher={Pearson Higher Education}
}

@misc{jclGithub,
author = "Kamran Zafar",
title = "Jar Class Loader Project Home GitHub",
year = "2015",
url = {https://github.com/kamranzafar/JCL/},
note = "Online; accessed 2015, 06 August"
}

@misc{MySQLRef,
author = "MySQL",
title = "MySQL 5.7 Reference Manual",
year = "2015",
url = {http://dev.mysql.com/doc/refman/5.7/en/data-types.html},
note = "Online; accessed 2015, December 24"
}

@misc{javaJarCommand,
author = "Oracle",
title = "Viewing the Contents of a JAR File",
year = "1995, 2015",
url = {https://docs.oracle.com/javase/tutorial/deployment/jar/view.html},
note = "Online; accessed 2016, January 12"
}

@misc{Jackson,
author = "FasterXML",
title = "Jackson Project Home GitHub",
year = "2015",
url = {https://github.com/FasterXML/jackson},
note = "Online; accessed 2016, January 19"
}

@misc{googleGson,
author = "googleGson",
title = "Gson Project Home GitHub",
year = "2015",
url = {https://github.com/google/gson},
note = "Online; accessed 2016, January 19"
```

```
}

@misc{seidlMetricsWeb,
author = "Richard Seidl",
title = "Modeling Metrics for UML Diagrams",
year = "2010",
url = {http://www.richard-seidl.com/modeling-metrics-for-uml-diagrams/},
note = "Online; accessed 2015, November 30"
}

@misc{aivostoProjectAnalyzer,
author = "Aivosto Oy",
title = "Project Analyzer v10.2",
year = "1994-2015",
url = {http://www.aivosto.com/},
note = "Online; accessed 2015, November 30"
}

@misc{metricsReloaded,
author = "Bas Leijdekkers",
title = "MetricsReloaded for IntelliJ IDEA",
year = "2011-2016",
url = {https://github.com/BasLeijdekkers/MetricsReloaded},
note = "Online; accessed 2016, January 31"
}

@misc{siemensGeorge2010,
author = "Siemens, George",
title = "What Are Learning Analytics?",
year = "2010",
url = {http://www.elearnspace.org/blog/2010/08/25/what-are-learning-analytics/},
note = "Online; accessed 2015, May 10"
}

@misc{sakaiLMSweb,
author = "Apereo Foundation",
title = "Sakai Open Source LMS",
year = "2014",
url = {https://sakaiproject.org/},
note = "Online; accessed 2015, May 10"
}


@misc{apereoHomePageIncubatorLAI,
title = {Apereo Home Page Incubator Project on Learning Analytics Initiative},
month = August,
year = {2015},
howpublished={\url{https://www.apereo.org/communities/learning-analytics-initiative}},
note = "Online; accessed 2016, February 04"
}
```

```
@misc{openContentWeb,
author = "Opencontent.org",
title = "Defining the Open in Open Content",
year = "1999",
url = {http://www.opencontent.org/definition/},
note = "Online; accessed 2016, February 04"
}


@misc{apacheMavenWeb,
author = "The Apache Software Foundation",
title = "Maven Introduction",
year = "2002-2016",
url = {https://maven.apache.org/what-is-maven.html},
note = "Online; accessed 2016, 10 February"
}


@misc{jitpackIoWeb,
author = "JitPack.io",
title = "JitPack, Publish JVM and Android libraries",
year = "2015-2016",
url = {https://jitpack.io/},
note = "Online; accessed 2016, 10 February"
}


@misc{gitHubWeb,
author = "GitHub, Inc.",
title = "GitHub",
year = "2016",
url = {https://github.com/},
note = "Online; accessed 2016, 19 February"
}
```