JOURNAL OF LEARNING ANALYTICS

SOLAR SOCIETY for LEARNING ANALYTICS RESEARCH

Volume 5(1), 92 – 100. http://dx.doi.org/10.18608/jla.2018.51.7

A Modular and Extensible Framework for Open Learning Analytics

Arham Muslim[1]*, Mohamed Amine Chatti[2], Muhammad Bassim Bashir[3], Oscal Eduardo Barrios Varela[4], Ulrik Schroeder[5]

Abstract

Open Learning Analytics (OLA) is an emerging concept in the field of Learning Analytics (LA). It deals with learning data collected from multiple environments and contexts, analyzed with a wide range of analytics methods to address the requirements of different stakeholders. Due to this diversity in different dimensions of OLA, the LA developers and researchers face numerous challenges while designing solutions for OLA. The Open Learning Analytics Platform (OpenLAP) is a framework that addresses these issues and lays the foundation for an ecosystem of OLA that aims at supporting learning and teaching in fragmented, diverse, and networked learning environments. It follows a user-centric approach to engage end users in flexible definition and dynamic generation of personalized indicators. In this paper, we address a subset of OLA challenges and present the conceptual and implementation details of the analytics framework component of OpenLAP, which follows a flexible architecture that allows the easy integration of new analytics methods and visualization techniques in OpenLAP to support end users in defining indicators based on their needs in order to embed the results into their personal learning environment.

Keywords

Personalized learning analytics, OpenLAP, analytics framework, modularity, extensibility.

Submitted: 07/06/17 — **Accepted:** 10/27/17 — **Published:** 04/09/18

**Corresponding author [1]Email: muslim@cil.rwth-aachen.de Address: RWTH Aachen University, Germany, Ahornstr. 55, 52074 Aachen, Germany*

[2]Email: mohamed.chatti@uni-due.de Address: University of Duisburg-Essen, Germany, Forsthausweg 2, 47057 Duisburg, Germany

[3]Email:bassim.bashir@rwth-aachen.de Address: RWTH Aachen University, Germany, Ahornstr. 55, 52074 Aachen, Germany

[4]Email: oscar.barrios@rwth-aachen.de Address: RWTH Aachen University, Germany, Ahornstr. 55, 52074 Aachen, Germany

[5]Email: schroeder@informatik.rwth-aachen.de Address: RWTH Aachen University, Germany, Ahornstr. 55, 52074 Aachen, Germany

1. Introduction

Learning Analytics (LA) incorporates the concepts of big data and data analytics to improve the learning and teaching processes of different stakeholders (Siemens et al., 2011). The term LA has been defined at the First International Conference on Learning Analytics and Knowledge (LAK '11) as "the measurement, collection, analysis, and reporting of data about learners and their contexts, for purposes of understanding and optimizing learning and the environments in which it occurs" (Siemens & Long, 2011). The technological boom in the last decade has revolutionized the education sector, moving the learning environment from traditional classrooms (e.g., learning management systems, virtual learning environments) toward more open, self-organized and networked settings (e.g., personal learning environments, massive open online courses; Chatti, 2010). This openness has led to the emergence of a new LA research area called Open Learning Analytics (OLA). In general, OLA encompasses different stakeholders associated with a common interest in LA but with diverse needs and objectives, a wide range of data coming from various learning environments and contexts, as well as multiple infrastructures and methods that enable us to draw value from data in order to gain insight into learning processes (Chatti, Muslim, & Schroeder, 2017).

In order to provide an effective solution, OLA introduces a number of challenges for LA practitioners, developers, and researchers. These include data aggregation and integration, interoperability, specifications and standards, reusability, modularity, flexibility and extensibility, performance and scalability, usability, privacy, transparency, and personalization (Chatti et al., 2017).

In this paper, we focus on the modularity and extensibility challenges in OLA and present the conceptual and implementation details of the analytics framework component of the Open Learning Analytics Platform (OpenLAP; Chatti et al., 2017). The main aim of the analytics framework is to provide a flexible architecture that makes it easy to extend OpenLAP with new analytics methods and visualization techniques to support end users in self-defining indicators according to their needs.

The remainder of the paper is structured as follows: In Section 2, we present the architecture and the main components of OpenLAP; Section 3 presents the design and implementation details of the analytics framework component of OpenLAP; finally, Section 4 concludes the paper and gives perspectives for future work.

2. Open Learning Analytics Platform (OpenLAP)

Chatti et al. (2017) propose a vision for an open learning analytics ecosystem and present a concrete conceptual and technical architecture for OpenLAP.[1] It provides end users with a user-centric mechanism to flexibly and dynamically generate their personalized indicators. In order to meet the requirements of diverse users, OpenLAP adapts a modular and extensible architecture that allows the easy integration of new analytics modules, analytics methods, and visualization techniques. In the following sections, we present a brief description of the OpenLAP abstract architecture and discuss supported system scenarios.

2.1. Abstract Architecture

The abstract architecture of OpenLAP shown in Figure 1 consists of three main components, namely *Data Collection and Management*, *Indicator Engine*, and *Analytics Framework*.

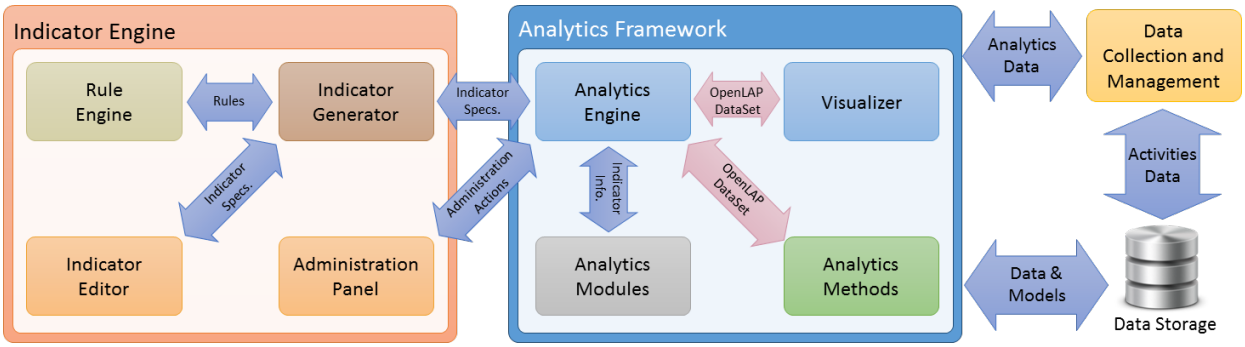

Figure 1. OpenLAP Abstract Architecture (Muslim, Chatti, Mughal, & Schroeder, 2017).

The *Data Collection and Management* component in OpenLAP is responsible for collecting learning activities data from different sources adhering to the privacy policies of OpenLAP and generating the learner and context models from it. OpenLAP uses the data model called Learning Context Data Model (LCDM) suggested by Thüs, Chatti, Greven, and Schroeder (2014) in the frame of the Learning Context Project.[2] LCDM represents a user-centric, modular, easy-to-understand data model that holds additional semantic information about the context in which an event has been generated (Lukarov et al., 2014). OpenLAP follows a rule-based approach to define the data access mechanism based on the selected parameters of the indicator. These rules can easily be modified to allow adaptation of other data models for LA, such as IMS Caliper, xAPI, Activity Streams, CAM, MOOCdb, or DiscourseDB (Muslim, Chatti, Mahapatra, & Schroeder, 2016). The *Data Collection and Management* component in OpenLAP provides an LCDM-based API that enables collecting data from various sources. For each new source, a data collection component (collector) needs to be developed. It can be an integrated component in a source that gathers data and pushes it to OpenLAP in the LCDM format. It can also be an intermediate component (adapter) that receives data from a source and transforms it into the LCDM format before sending it to OpenLAP (Chatti et al., 2017).

The aim of the *Indicator Engine* in OpenLAP is to achieve personalized and goal-oriented LA by following a Goal–Question–Indicator (GQI) approach that allows users to easily define new indicators through an interactive UI. Additionally, it provides an administration panel to manage the analytics modules, analytics methods, and visualization techniques in OpenLAP (Muslim, Chatti, Mughal, & Schroeder, 2017).

[1] http://lanzarote.informatik.rwth-aachen.de/openlap
[2] http://www.learning-context.de/

The *Analytics Framework* is the core component of OpenLAP, which manages, generates, and executes indicators. It is responsible for fetching raw data from the database, performing the analysis and visualizing the indicator. Additionally, the *Analytics Framework* implements the extensibility by supporting easy mechanisms to manage, add, and remove analytics modules, analytics methods, and visualization techniques in OpenLAP. It is the combination of five sub-components; namely, *OpenLAP-DataSet*, *Analytics Modules*, *Analytics Methods*, *Visualizer*, and *Analytics Engine*.

2.2. System Scenarios

The different components in OpenLAP interact with each other to provide two main system scenarios, namely the *indicator generation* and the *indicator execution*.

During the indicator generation, the user interacts with the *Indicator Engine* UI to define an LA goal, ask an LA question, and associate multiple indicators to answer this question. In order to define a new indicator, the user follows a 4-step process. First, select an appropriate dataset by exploring the learning events data. Second, apply various filters on the selected dataset. Third, choose an analytics method and maps the dataset columns to the inputs of the analytics method to analyze the dataset. Fourth, select an appropriate visualization technique, map the outputs of the analytics method to the inputs of the visualization technique, and preview the indicator visualization. After finalizing, the indicator is validated by the *Analytics Engine* and saved in the *Analytics Modules* as a triad containing references to the indicator query, the chosen analytics method, and the visualization technique. Additionally, the triad contains the two mappings: query-method and method-visualization. In return, the user gets an HTML and JavaScript based indicator request code for this indicator containing the triad identifier. This indicator request code can then be embedded in any client application (e.g., any Web page, dashboard, LMS) where it will visualize the indicator.

The indicator execution scenario is initiated when the indicator request code embedded in the client application communicates with OpenLAP to visualize the indicator. The *Analytics Engine* intercepts the request and validates it. Next, it gets the triad from the respective *Analytics Module* using the triad identifier in the request. Afterwards, it gets the related query from the database, executes it to get the raw data, and transforms it to the *OpenLAP-DataSet*. The *OpenLAP-DataSet* and the mapping query-method is sent to the analytics method referenced in the triad for analysis. The received analyzed data as an *OpenLAP-DataSet* and the mapping method-visualization is forwarded to the visualization technique referenced in the triad. The *Visualizer* generates the indicator visualization code and returns it to the *Analytics Engine*, which forwards it to the requesting client application to visualize the indicator.

3. Analytics Framework in OpenLAP

In this paper, we focus on the *Analytics Framework* component of OpenLAP, which is responsible for implementing modularity and extensibility by providing a flexible infrastructure that enables us to easily integrate new analytics methods and visualization techniques into OpenLAP. In the following sections, we present the *Analytics Framework* by discussing one of the possible user scenarios, requirements, and implementation details.

3.1. User Scenario

Asma is a researcher at XYZ University, which uses OpenLAP to support open learning analytics. Asma developed a mobile application for uploading learning materials to her course in the LMS. She is interested in using the *Analytics Framework* of OpenLAP to analyze which of the learning materials are most viewed. She uses the *Indicator Engine* UI of OpenLAP to define a new indicator called "Top 10 Learning Material," which should apply a "Count top 10 items" analytics method and visualize it using the "Bar Chart" format of the "Google Charts" library. Unfortunately, neither "Count top 10 items" nor "Google Charts" are available in the *Analytics Framework*. Thus, she develops a new analytics method called "Count top 10 items" by following the provided templates and guidelines and uploads it to the *Analytics Framework* using the administration panel in the *Indicator Engine*. Furthermore, she develops a new visualization method called "Bar Chart" for "Google Charts" and uploads it to the *Analytics Framework*. Asma goes back to the *Indicator Engine* UI and selects the newly added analytics method and visualization technique to be applied in the indicator.

3.2. Requirements

Developing a framework that allows for the dynamic addition of new analytics methods and visualization techniques at runtime as well as managing a growing number of user-defined indicators is a challenging task. Therefore, a modular, service-oriented approach should be followed in the design of the *Analytics Framework* to support easy, effective communication between loosely coupled modules. Additionally, the framework should be flexible and extensible to support a growing amount of analytics functionality by enabling the smooth plug-in of new modules, analytics methods, and visualization techniques after the platform has been deployed and is running. These newly added analytics methods and visualization techniques should be

reusable so that other users can also define their personalized indicators. These requirements represent a subset of the OpenLAP requirements, which are discussed in more detail in Chatti et al. (2017).

3.3. Implementation

The *Analytics Framework* consists of four main components: *Analytics Modules*, *Analytics Methods*, *Visualizer,* and *Analytics Engine* as well as a data exchange format called *OpenLAP-DataSet,* as shown in Figure 1. It is a Java Spring Framework[3] based web application in which each component follows the Facade design pattern (Gamma, Helm, Johnson, & Vlissides, 1994) and exposes API endpoints through a single simplified interface called "Controller" to communicate with other components. For the sake of understandability, API endpoints of each component are grouped together based on the nature of the task performed by them, as shown in the technical architecture of the *Analytics Framework* in Figure 2. In the following sections, we discuss the implementation details of the *Analytics Framework* at a level of detail that enables insights into the modularity and extensibility mechanisms in OpenLAP. More details about the functionalities of each component in the *Analytics Framework* are available on the project GitHub wiki.[4]

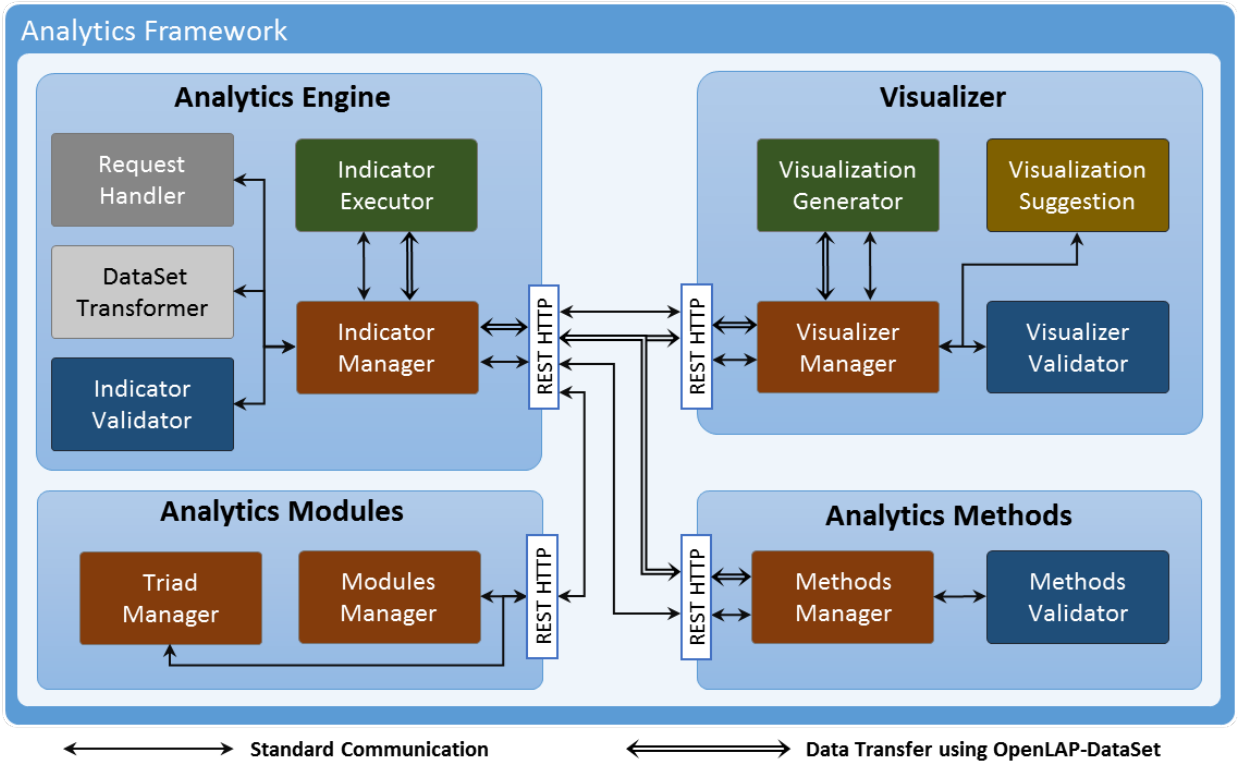

Figure 2. Technical architecture of the analytics framework.

3.3.1. OpenLAP-DataSet

The internal data exchange format used in the *Analytics Framework is* the *OpenLAP-DataSet*. It is a modular JSON-based serializable dataset to validate and exchange data between different components. Since the modular approach is used to develop the *Analytics Framework*, different components act with relative independence from each other. Thus, a data exchange model is needed that can easily be serialized to and from JSON and allow automatic parsing.

The *OpenLAP-DataSet* is a collection of "OpenLAP-DataColumns," as shown in Figure 3. Each column consists of two distinct sections, namely an "OpenLAP-ColumnConfigData" section containing metadata for describing the column ID, type and required flag, and an "OpenLAP-Data" section that stores the data itself.

[3] http://projects.spring.io/spring-framework/

[4] https://github.com/OpenLearningAnalyticsPlatform/

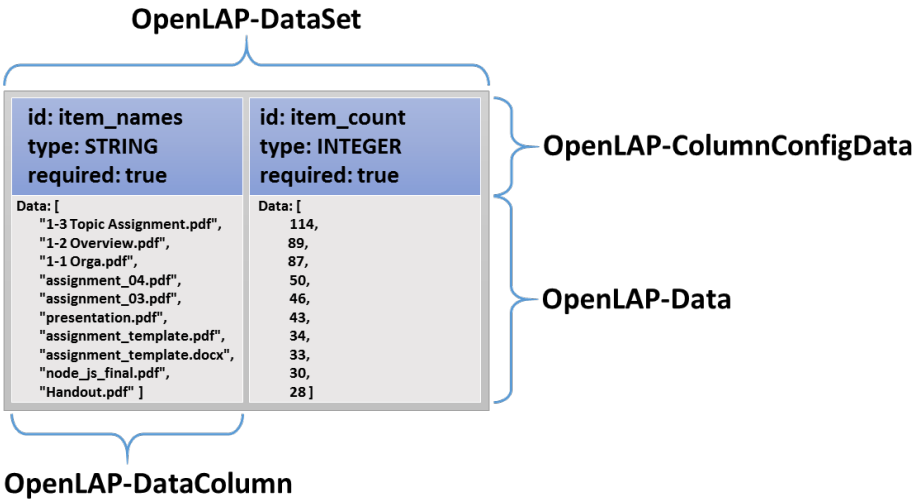

Figure 3. OpenLAP-DataSet abstract architecture.

In order to provide support for dynamic validation of types and the presence of all required columns in the *OpenLAP-DataSet* before sending it to another component, a class named "OpenLAP-PortConfig" is used. This class specifies which columns of the sending component's *OpenLAP-DataSet* should map to which columns of the receiving component's *OpenLAP-DataSet*. The sending component generates this configuration and sends it to the receiving component for validation. Since the configuration only has the metadata section of the *OpenLAP-DataSet* (i.e., "OpenLAP-ColumnConfigData"), it is relatively lightweight. The receiving component executes the "validateConfiguration()" method of its *OpenLAP-DataSet* with the received configuration and returns the compatibility results to the sending component. After validation, the configuration along with the *OpenLAP-DataSet* including the data is sent to the receiving component, which then replaces the incoming *OpenLAP-DataSet* metadata with the metadata of its *OpenLAP-DataSet* using the configuration in order to process it.

3.3.2. Analytics Modules

The component representing a collection in which each module corresponds to an analytics goal such as monitoring, personalization, prediction, assessment, reflection, and recommendation is represented by the *Analytics Modules*. Each module is responsible for managing a list of analytics methods associated with each goal. Moreover, each module manages a list of user-defined indicators in the form of triads. As shown in Figure 2, the *Analytics Modules* component consists of two groups: "Modules Manager" and "Triad Manager." It is only accessed by the *Analytics Engine*. Thus, all communication between the *Analytics Modules* and other components goes through the *Analytics Engine*.

Modules Manager. Methods to manage the analytics goals and the list containing metadata of the associated analytics methods is provided by the "Modules Manager." To access the list of analytics goals available in the *Analytics Modules*, methods such as "getAnalyticsGoal()" and "getAllAnalyticsGoals()" are available; these are used to help end users in selecting appropriate analytics goals during the process of indicator definition in the *Indicator Engine* UI. If the user does not find the required analytics goal, they can request a new one, which is created using "addAnalyticsGoal()." To moderate the creation of new analytics goals, all newly created goals have an internal flag marked as "inactive" and are not accessible by users until they are activated by OpenLAP system administrators via the administration panel in the *Indicator Engine*, using "activateAnalyticsGoal()."

During the process of defining new indicators, the user can select an analytics method to apply before visualizing the indicator. The list of all analytics methods available in the *Analytics Framework* is presented to the user in which the analytics methods previously used in conjunction with the selected goal are provided on top of the list using "getAllAnalyticsMethodsOfGoal()." If the user selects an analytics method not previously used with the selected goal, that analytics method will be associated with this goal using "addAnalyticsMethodToGoal()."

Triad Manager. Basic methods to access and save triads in the Analytics Modules are provided by the "Triad Manager" group. A Triad in the context of the Analytics Framework is defined as a data structure that represents a single user-defined indicator. It contains references to the indicator query, to the associated analytics method, and to the visualization technique to be used for the indicator (Chatti et al., 2017). Additionally, the triad also stores two "OpenLAP-PortConfig" configurations (see Section 3.3.1); the first defines the mapping between the OpenLAP-DataSet of the query data and the input OpenLAP-

DataSet of the associated analytics method; the second defines the mapping between the output OpenLAP-DataSet of the analytics method and the input OpenLAP-DataSet of the selected visualization technique.

3.3.3. Analytics Methods

The component responsible for managing the repository of all available analytics methods in the *Analytics Framework* is *Analytics Methods*. Analytics methods of any type can be added to the repository, such as statistics, data mining (DM), and social network analysis (SNA). As shown in Figure 2, *Analytics Methods* consists of two groups: "Methods Manager" and "Methods Validator." Similar to the *Analytics Modules*, the *Analytics Methods* also communicates with other components of the *Analytics Framework* through the *Analytics Engine*.

Methods Manager. The "Methods Manager" provides methods for managing available analytics methods in the *Analytics Framework,* such as "viewAllAnalyticsMethods()," and "updateAnalyticsMethod()." During the indicator generation process, the *Analytics Engine* uses the methods "getInputPorts()" and "getOutputPorts()" of the selected analytics method to get the expected input and output *OpenLAP-DataSet*s metadata. The *Indicator Engine* UI uses this metadata to support end users in defining the mappings query-method and method-visualization and generate their respective "OpenLAP-PortConfig."

In order to implement a new analytics method, developers perform the following steps to extend the "<<Abstract>>AnalyticsMethod" class:

- Initialize the expected input and output *OpenLAP-DataSet*s in the constructor.
- Override the "implementationExecution()" method that should perform the analysis on the input *OpenLAP-DataSet* and store the analysis results in the output *OpenLAP-DataSet*.
- If the analytics method is predictive, then the result of the training phase of the machine learning algorithm should be provided as an XML-based predictive model using Predictive Model Markup Language (PMML; Guazzelli, Zeller, Lin, & Williams, 2009). The "hasPMML()" method should return true and the "getPMMLInputStream()" method should provide input stream to read the XML file.

The developer can then add the new analytics method to the *Analytics Methods* repository via the administration panel in the *Indicator Engine*. Besides uploading the JAR bundle that contains the compiled relevant files, the developer needs to specify the name and description of the analytics method, name of the developer, and the name of the class that implements the "<<Abstract>>AnalyticsMethod" class. Internally, the *Analytics Engine* uses the provided information to create a JSON object and calls the "uploadAnalyticsMethod()" method with the JAR file and the JSON object as parameters to add the new analytics method to the *Analytics Methods* repository. A more detailed step-by-step guide on how to implement a new analytics method along with a concrete example is available on the project GitHub wiki.[5]

Methods Validator. The "validateNewAnalyticsMethod()" — provided by the "Methods Validator" — checks before adding them to the collection of available analytics methods in the *Analytics Framework*. The method checks whether the class specified in the JSON object has implemented the "<<Abstract>>AnalyticsMethod" class and verifies if it has all the required methods. It also verifies the uniqueness of the analytics method name and file name in order to avoid conflicts. If a PMML file is also included in the JAR bundle, then it is also validated. After successful validation, the information provided in the JSON object is stored in the internal database and the new analytics method is made available in the *Analytics Framework*.

3.3.4. Visualizer

Providing an extensible and modular architecture for managing visualization techniques in the *Analytics Framework* is the responsibility of the *Visualizer* component. A visualization technique consists of a visualization framework, such as Google Charts, D3/D4, jpGraph, Dygraphs, and jqPlot, along with their supported visualization types, such as bar chart, pie chart, or line chart. The *Visualizer* is also responsible for generating the indicator visualization code consisting of HTML and JavaScripts that can be embedded easily into any client application. As shown in Figure 2, the *Visualizer* consists of four groups: "Visualizer Manager," "Visualization Generator," "Visualizer Validator," and "Visualization Suggestion."

Visualizer Manager. The "Visualizer Manager" provides methods to manage visualization frameworks and their supported visualization types. These methods are used by the *Analytics Engine* via the administration panel in the *Indicator Engine*. In order to add a new visualization framework in the *Visualizer*, the following tasks need to be performed for each visualization type in the framework:

- Extend the "<<Abstract>>VisualizationCodeGenerator" class.
- Initialize the expected input *OpenLAP-DataSet* in the "initializeDataSetConfiguration()" method.

[5] https://github.com/OpenLearningAnalyticsPlatform/OpenLAP-AnalyticsMethodsFramework

- As each visualization type of a framework may require the data to be present in a different structure, an interface called "DataTransformer" is provided, which can be extended to transform the data from the expected input *OpenLAP-DataSet* into the expected data structure required by the visualization type. The same "DataTransformer" can be reused by multiple visualization types if they expect the same data structure.
- Override the "visualizationCode()" method and add the logic to generate the indicator visualization code using the transformed data generated in the previous step.

The developer then needs to create a bundled JAR file containing the implementation classes of the new visualization technique and upload it to the *Visualizer* via the administration panel in the *Indicator Engine* along with further information, including the visualization framework name and description, developer name, and the list of the implemented visualization types with the used data transformers. The JAR file and the specified metadata are used by the *Analytics Engine* as parameters of the method "uploadVisualizationFramework()" to add the new visualization technique to the *Visualizer*. A more detailed description on how to add a new visualization framework is available on the project GitHub wiki.[6]

Visualizer Validator. The "Visualizer Validator" provides a "validateFramework()" method to validate the new visualization framework before making it available in the *Analytics Framework*. As soon as a new framework is uploaded to the *Visualizer* at runtime, first and foremost the JSON object is checked. The implemented classes stated as part of the visualization methods and data transformers in the metadata are loaded by a class loader to check if they implement the correct interface/abstract classes. After a successful validation, the actual upload of the frameworks starts. This step includes storing the JAR in the deployment server's file system and making the new visualization framework available.

Visualization Generator. The "Visualization Generator" provides a "getIndicatorVisualizationCode()" method to get the indicator visualization code by providing the visualization framework name/id, the visualization type name/id, the output *OpenLAP-DataSet* of the analytics method, and the mapping method-visualization. This method first maps the output *OpenLAP-DataSet* of the analytics method to the expected input *OpenLAP-DataSet* of the visualization technique. It then calls the "DataTransformer" provided in the JSON object to transform the expected input *OpenLAP-DataSet* into the expected data structure required by the specified visualization type. After that, the "visualizationCode()" method in the implemented "VisualizationCodeGenerator" class is called along with the transformed data to generate the indicator visualization code, which is sent back to the client where it is visualized. The indicator visualization code for the "Top 10 Learning Material" indicator is shown below and its visualization on the client side is shown in Figure 4.

```
<div id="chartdiv"></div>
<script type="text/javascript">
var data=google.visualization.arrayToDataTable([
        ["x-axis_labels," "y-axis_values"], ["1-3 Topic Assignment.pdf," 114],
        ["1-2 Overview.pdf," 89], ["1-1 Orga.pdf," 87], ["assignment_04.pdf," 50],
        ["assignment_03.pdf," 46], ["presentation.pdf," 43], ["assignment_template.pdf," 34],
        ["assignment_template.docx," 33], ["node_js_final.pdf," 30], ["Handout.pdf," 28] ] );
var options = { title: "Top 10 Learning Material," width: 440, height: 200, is3D: true,
        chartArea:{ width: "95%," height: "135," left: "50," top: "10" }, vAxis:{ title: "Count" },
        hAxis:{ title: "Learning Materials" }, backgroundColor: { fill: "transparent" } };
var chart = new google.visualization.ColumnChart(document.getElementById("chartdiv"));
chart.draw(data, options);
</script>
```

Visualization Suggestion. The "Visualization Suggestion" provides methods to suggest which possible visualization techniques can be applied based on the OpenLAP-DataSet metadata. The suggestions are generated by comparing a given OpenLAP-DataSet metadata with the expected input OpenLAP-DataSet metadata of each visualization type stored in the Visualizer.

3.3.5. Analytics Engine

The Analytics Engine is the main orchestrator of the Analytics Framework. It is responsible for managing the indicator generation and the indicator execution processes (see Section 2.2). As shown in Figure 2, the Analytics Engine consists of five groups: "Request Handler," "Indicator Manager," "Indicator Validator," "DataSet Transformer," and "Indicator Executor."

[6] https://github.com/OpenLearningAnalyticsPlatform/OpenLAP-Visualizer-Framework

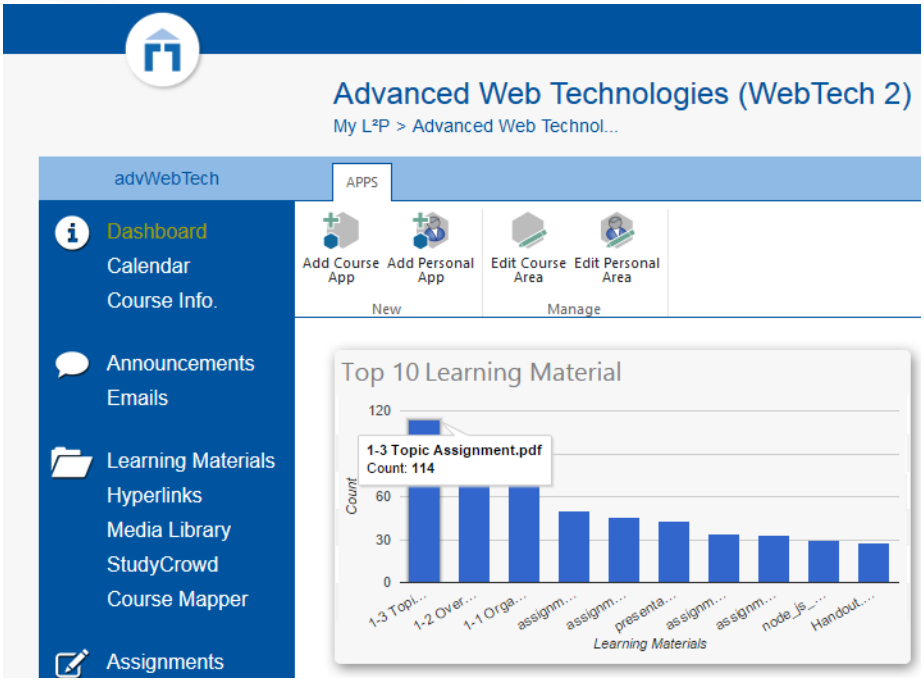

Figure 4. Visualization of "Top 10 Learning Material" indicator**.**

Request Handler. The "Request Handler" group is composed of methods responsible for handling the incoming requests from the *Indicator Engine* to generate new indicators and from external systems to execute indicators. The methods "getIndicatorsByQuestion()," "getIndicatorsByGoal()," "getIndicatorByID()," and "getAllIndicators()" are mainly used by the *Indicator Engine* to implement the GQI approach in the indicator generation process to get an indicator request code consisting of HTML and JavaScripts that can be embedded in any client application. The "getIndicatorData()" method is called by the indicator request code from a client application in the indicator execution process to get the indicator visualization code (see Section 2).

Indicator Manager. The "Indicator Manager" group consists of methods for accessing and saving triads to the Analytics Modules. After finalizing the new indicator generation process in the Indicator Engine, the Analytics Engine saves the indicator as a triad using the "saveTriad()" method. Whenever the "getIndicatorData()" method is called from a client application with the triad ID as parameter, the Analytics Engine starts the indicator execution process by getting the triad from the Analytics Modules using the "getTriad()" method that contains references to the indicator query, the analytics method, and the visualization technique, as well as the "query-method" and "method-visualization" mappings.

Indicator Validator. The "Indicator Validator" provides the "validateIndicator()" method that is called at the end of the indicator generation process to determine whether the mappings indicator query-method and method-visualization are valid or not using the "validateConfiguration()" method available in the OpenLAP-DataSet of the chosen analytics method and the visualization technique, respectively.

DataSet Transformer. The "DataSet Transformer" group provides a method to convert the indicator query result data from the underlying database format to the OpenLAP-DataSet. Currently, an SQL-based database is being used by the Analytics Framework to store learning activities data using LCDM data model. Thus, only a "convertSQLToDataSet()" method is provided, which converts the SQL-based data to OpenLAP-DataSet. This provides the Analytics Framework with the ability to easily adapt different underlying databases by implementing new "DataSet Transformer" methods for the new databases (e.g., NoSQL databases).

Indicator Executor.The "Indicator Executor" group provides two methods. The "executeTriad()" method is used to perform the indicator execution process including data acquisition, transformation, analysis, and visualization. The "executeIndicatorForPreview()" method is called to preview the result of the indicator generation process. Thereby only a subset of the indicator query data is used so that the visualization is generated relatively faster. The methods in this group also handle the exceptions that might occur during the indicator execution process.

4. CONCLUSION AND FUTURE WORK

Open learning analytics (OLA) is a particularly rich area for future research. In this paper, we addressed the modularity and extensibility challenges in OLA by presenting the conceptual and implementation details of the analytics framework component in the Open Learning Analytics Platform (OpenLAP). The analytics framework lays the foundation of a simple yet powerful framework that provides an easy, flexible mechanism for LA researchers and developers to add new analytics methods and visualization techniques to OpenLAP by following the provided templates and guidelines. Our future work includes implementing further analytics methods based on data mining and social network analysis algorithms as well as new visualization techniques using, for example, D3.js, C3, Highcharts, and InfoVis. Additionally, the current implementation of the administration panel only allows for the adding of new analytics methods and visualization techniques by researchers and developers. Therefore, an enhancement of the administration panel is planned to allow easy modification, deletion, and management of the added analytics methods and visualization techniques. In order to support developers in validating their newly developed analytics methods and visualization techniques, we have plan to provide a mechanism in the administration panel to generate sample data based on the inputs of the developed component and a set of guidelines to use it. The sample data can be used to debug and validate components before uploading them to OpenLAP. In terms of evaluation, we are preparing to conduct the pilot phase for OpenLAP at RWTH Aachen University and the University of Duisburg-Essen in which we plan to evaluate the architecture quality of OpenLAP, the extensibility process, and the usability and usefulness of the indicator generation process.

Declaration of conflicting interest

The author(s) declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

Funding

The author(s) declared no financial support for the research, authorship, and/or publication of this article.

References

Chatti, M. A. (2010). *Personalization in technology enhanced learning: A social software perspective.* Dissertation RWTH Aachen, Shaker Verlag. http://www.shaker.eu/shop/978-3-8322-9575-2

Chatti, M. A., Muslim, A., & Schroeder, U. (2017). Toward an open learning analytics ecosystem. In B. K. Daniel (Ed.), *Big data and learning analytics in higher education* (pp. 195–219). Springer. http://dx.doi.org/10.1007/978-3-319-06520-5_12

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). *Design patterns: Elements of reusable object-oriented software.* London: Pearson Education.

Guazzelli, A., Zeller, M., Lin, W.-C., & Williams, G. (2009). PMML: An open standard for sharing models. *The R Journal, 1*, 60–65. https://journal.r-project.org/archive/2009/RJ-2009-010/index.html

Lukarov, V., Chatti, M. A., Thüs, H., Kia, F. S., Muslim, A., Greven, C., & Schroeder, U. (2014). Data models in learning analytics. *Proceedings of DeLFI Workshops*, 15 September 2014, Freiburg, Germany (pp. 88–95). http://ceur-ws.org/Vol-1227/paper22.pdf

Muslim, A., Chatti, M. A., Mahapatra, T., & Schroeder, U. (2016). A rule-based indicator definition tool for personalized learning analytics. *Proceedings of the 6th International Conference on Learning Analytics and Knowledge* (LAK '16), 25–29 April 2016, Edinburgh, UK (pp. 264–273). New York: ACM. http://dx.doi.org/10.1145/2883851.2883921

Muslim, A., Chatti, M. A., Mughal, M., & Schroeder, U. (2017). The goal–question–indicator approach for personalized learning analytics. *Proceedings of the 9th International Conference on Computer Supported Education* (CSEDU 2017) 21–23 April 2017, Porto, Portugal (Vol. 1, pp. 371–378). ScitePress. http://dx.doi.org/10.5220/0006319803710378

Siemens, G., & Long, P. (2011). Penetrating the fog: Analytics in learning and education. *EDUCAUSE Review, 46*, 30. https://eric.ed.gov/?id=EJ950794

Siemens, G., Gasevic, D., Haythornthwaite, C., Dawson, S., Shum, S. B., Ferguson, R., . . . Baker, R. S. (2011). Open learning analytics: An integrated and modularized platform. *Proposal to design, implement and evaluate an open platform to integrate heterogeneous learning analytics techniques.* http://solaresearch.org/wp-content/uploads/2011/12/OpenLearningAnalytics.pdf

Thüs, H., Chatti, M. A., Greven, C., & Schroeder, U. (2014). Kontexterfassung, -modellierung und-auswertung in Lernumgebungen. *DeLFI 2014-Die 12. e-Learning Fachtagung Informatik* (pp. 157–162). Gesellschaft für Informatik. http://cs.emis.de/LNI/Proceedings/Proceedings233/157.pdf