A Rule-Based Indicator Definition Tool
For Personalized Learning Analytics

Master-Thesis

presented by
Mahapatra, Tanmaya
340959

First examiner: Prof. Dr. Ulrik Schroeder
Second examiner: PD Dr. Christoph Quix
Supervisors: Dr. Mohamed Amine Chatti, MSc. Arham Muslim

Aachen, September 14, 2015
Erklärung

Hiermit versichere ich, dass ich diese Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

I hereby declare that I have created this work completely on my own and used no other sources or tools than the ones listed.

Aachen, September 14, 2015
Tanmaya Mahapatra
Acknowledgement

Firstly, I would like to invoke the blessings of the Almighty without whose grace nothing is permissible in this world.

I would like to express my sincere gratitude from the very inner core of my heart to Dr. Ulrik Schroeder for providing an unique opportunity and all facilities for completing my thesis work in his supervision. I would like to thank Dr. Mohamed Amine Chatti for providing me with positive criticism throughout the work which played a pivotal role in successful completion of this thesis. I am extremely indebted to him for instilling in me the level of confidence and showing everything is achievable with perseverance.

My special thanks to Arham Muslim whose words became my sole source of inspiration to complete the work in time. His constant encouragement, positive criticism helped me progress in the right direction. I would also like convey my appreciation to Hendrik Thüs for helping me with database issues in the initial days of my work.

I am also indebted to my parents for their continuous support throughout the journey. My special thanks to all my friends especially Ahmed Mashlique Raihan, Devashish Jasani, Oleksandr Sabov, Petr Sergeevich Tarasenko, Sauvik Bhattacharya and Thomas Eck who made me feel that I was not alone at any stage of my work.

Last but not the least I am also grateful to all the members of the Department for their guidance and suggestions which ultimately culminated in the form of this thesis.
Abstract

Learning Analytics is being widely used to assist the improvement of teaching and learning process. The teaching and learning methodologies have drastically evolved with time i.e from traditional class room based teaching to modern day technology enhanced learning. In such a scenario constant monitoring, feedback and analysis of the process is of paramount importance.

Learning Analytics systems have been successfully used in real time scenarios to assist the teachers to improve the quality of education, identify at-risk students and enhance the student success rate. But still these systems lack in terms of user involvement and level of personalization which they offer to the end user. The user is often forced to use only the analytics and indicators available in the system. These indicators may not suit the need of all users.

This thesis aims to eliminate this problem of inflexibility by facilitating the user to define his own Question and Indicator. Hence there is an active participation of the user in the system. The conceptual approach to implement such a system and the development of the system has been the main work.

Finally, the work evaluates the system. From the results of the user evaluation, further development plans for this approach has been proposed. The system can be further enhanced in the light of this work thereby leading to a rapid scale personalization.
Chapter Overview

This document is divided into several chapters. Each chapter deals with a unique area related to the development of the system. Every chapter starts with a small overview thereby helping the user to have a gist of the chapter instead of going through the whole content and mostly ends either with a brief summary indicating where the next chapter is proceeding to. The various chapters contained in the document are listed below:

Introduction

The Introduction chapter exclusively introduces the problem of today’s education system and how Learning Analytics is actively being used to improve the teaching as well as learning process. It also explains the limitations of the traditional Learning Analytics systems and gives a brief overview of the objectives which the thesis work intends to achieve. Exposure to the research questions which the thesis will address has also been given.

Related Work

In this chapter, some of the existing Learning Analytics systems have been considered. Their working and limitations have been discussed.

Conceptual Approach

In this chapter, the conceptual approach of the Indicator Engine tool has been discussed along with relevant graphical illustrations. It is hoped that the user will get a feel of the overall working concept of the system.

Implementation

This chapter deals with the abstract architecture of the system. It also includes the various data models which the system uses to store user generated data like Questions and Indicators at run-time. It also describes the different components of the system and gives an overview of the implementation details including interesting code snippets as well as stating the technologies employed to achieve them. The Question-Indicator Editor is the core component of the system. The user interacts with this to create new Questions and Indicators at system run-time. All the
features offered by this editor as well all its sub-components which are working collaboratively have been explained.

Evaluation

Evaluation is the only true basis for justification. Therefore, in a dedicated chapter the entire evaluation process is explained. Evaluation mainly focussed on user expectation, usability and usefulness of the system. Each of these are discussed in this chapter.

Conclusion and Future Work

In this chapter, outcome of research questions are discussed, specially what are the verdict for several hypotheses. It would give a feel to the readers how the work justifies that the research questions stated initially were met. Lastly, the entire thesis work is summarized starting from the first milestone to the last milestone. It also throws light on what has been achieved with this thesis and also lists the possibilities of future work which can enhance the working of the system.

Appendix

A collection of interesting source codes, helpful information regarding various Entity-Relationship models employed by the system, user evaluation questionnaire and results are provided here.
# Contents

1 Introduction 1
   1.1 Motivation .................................................. 2
   1.2 Objectives ................................................. 3
   1.3 Research Questions ...................................... 3
   1.4 Outline ..................................................... 3

2 Related Work 5
   2.1 Learning Analytics (LA) ................................... 5
      2.1.1 LA Reference Model .................................. 6
      2.1.2 Process of LA ......................................... 7
      2.1.3 Open Learning Analytics .............................. 8
      2.1.4 Open Learning Analytics Platform (OLAP) ......... 9
   2.2 Existing LA Systems ...................................... 12
      2.2.1 Academic Analytics Tool (AAT) ........................ 12
      2.2.2 LeMo .................................................. 12
      2.2.3 Explanatory Learning Analytics Toolkit (eLAT) .... 13
      2.2.4 Workbench ............................................ 15
      2.2.5 Portable Learning Analytics .......................... 15
   2.3 Limitations .................................................. 16
   2.4 Summary ..................................................... 17

3 Conceptual Approach 19
   3.1 Scope and Requirements ................................. 19
   3.2 Abstract Architecture .................................... 20
   3.3 Working Scenario ......................................... 21
      3.3.1 Data Source .......................................... 21
      3.3.2 System and its functionalities ...................... 22
      3.3.3 Overview of User and System Interaction .......... 23
      3.3.4 Indicator Generation ................................ 24
   3.4 Use Case Scenarios ....................................... 24
      3.4.1 General Functionalities Use Case .................... 24
      3.4.2 Indicator Definition Use Case ....................... 25
      3.4.3 Filter Addition to Indicators Use Case ............. 26
      3.4.4 Indicator Modification Use Case .................... 27
   3.5 Summary ..................................................... 28
4 Implementation

4.1 Introduction ................................................. 29
4.2 Technologies .................................................. 29
4.3 Architecture ................................................... 30
   4.3.1 Indicator Engine Dispatcher Servlet (IEDS) .............. 30
   4.3.2 URL Handler Mapping ..................................... 31
   4.3.3 Spring MVC Controller .................................... 32
   4.3.4 Models .................................................. 32
   4.3.5 Views .................................................. 32
   4.3.6 View Resolver .......................................... 33
   4.3.7 CEWOLF Graphing Servlet ................................ 33
   4.3.8 Accessory Beans Running in Application Context ........ 34
4.4 Data Models .................................................. 34
   4.4.1 System User ER Model .................................... 34
   4.4.2 Question & Indicator ER Model ............................ 36
   4.4.3 Data Source ER Model .................................... 37
4.5 Activity Scenario: Interaction of Different Components .......... 37
4.6 Object Relational Mappings .................................... 40
4.7 Controllers ................................................... 41
4.8 Parsing & Query Generation .................................... 42
   4.8.1 Data Source and Filter Specifications ....................... 43
   4.8.2 Parsing and Query Generation with Drools ................. 43
      4.8.2.1 Operation Number Processor ............................ 44
      4.8.2.2 Drools Rule File ....................................... 44
      4.8.2.3 Parser Methods ........................................ 45
4.9 User Interface & Client side Technology .......................... 47
4.10 Miscellaneous Components ..................................... 47
4.11 The Question-Indicator Editor (QI Editor) ......................... 50
4.12 QI Editor Components ........................................ 51
   4.12.1 Question Information ..................................... 52
   4.12.2 Question Summary ........................................ 52
   4.12.3 Indicator Information ..................................... 53
   4.12.4 Indicator Properties & Summary ............................ 53
4.13 Summary ..................................................... 55

5 Evaluation ....................................................... 57

5.1 Evaluation Questionnaire ....................................... 57
   5.1.1 User Background Evaluation ............................... 58
   5.1.2 User Expectations Evaluation .............................. 58
   5.1.3 System Usefulness Evaluation .............................. 58
   5.1.4 Usability Evaluation ....................................... 60
      5.1.4.1 Usability of a System .................................... 60
      5.1.4.2 The System Usability Scale (SUS) ....................... 61
      5.1.4.3 Score Interpretation with SUS ............................ 62
      5.1.4.4 Usability Evaluation Results ............................ 62
5.2 Evaluation Results ............................................. 63
1 Introduction

“The mere imparting of information is not education.”

–Carter G. Woodson

This chapter describes the way educational process has evolved over the ages. The process of imparting education has evolved over the ages and the rapid application of technologies to continuously improve as well as adapt the process to the growing needs of the society. Teaching alone does not culminate in successful transfer of knowledge. Interaction and dynamic feedback are of paramount importance for the process to be successful and benefit the society in long run. Participants on both sides i.e teachers as well as students should stay tuned to one another and leverage maximum benefit from the process. Once upon a time classroom based interaction and direct feedback sufficed the needs. But with the advent of e-learning technologies the scenario has changed dramatically thereby amplifying the severity of the issue. Teachers stand clueless about the progress of students, identifying at-risk students or simply having an idea of how much of the content delivered has been assimilated by the students. Learning Analytics (LA) is being actively employed to tackle this issue with varying degrees of success.

Education is considered as the key element of a knowledge based and innovation driven economy [48]. Today’s e-learning Model and approach is heavily influenced by the traditional postal correspondence based distance education. Imparting distance education also requires close monitoring and behavioural patterns of the students just like in the case of regular education. Here in RWTH, the main lectures are being provided in a regular fashion (class room based ) where the teacher and the students have a face to face discussion with respect to the literature being taught. But the assignment distribution, submission, additional literature references and various other activities are being done using a learning management system (LMS) called L²P. The feedback from such a system is very limited. The teachers need a tool to “monitor” the students as well as the process. The students need a tool for “self-reflection” and “awareness”. So here LA comes into play.
1.1 Motivation

LA makes use of a wide range of research fields and generally deals with the development of methods that harness educational data sets to support the learning process [13]. LA has the potential to impact the current model of education [59] and helps in answering possible questions arising in the minds of teachers and students. For example the general questions asked by teachers have been broadly classified into the following major categories [12]:

- Qualitative questions
  1. How do students like/rate/value specific learning offerings?
  2. How difficult/easy is it to use the learning offering?
- Quantitative questions
  1. How often do students use a learning environment?
  2. Are there specific learning offerings that are not used at all?
- Questions dealing with data correlation
  1. How many (percent of the) learning modules are student viewing?
  2. Which didactical activities facilitate continuous learning?

LA deals with collection and analysis of traces that users leave online. Teachers and learners can benefit differently from LA. A teacher could observe live statistics about what and how learners are doing and based on this intervene and adapt the course when required. For example, a LA-based notification system can predict which students are about to drop out [65].

A dashboard is a visual display of the most important information needed to achieve one or more learning objectives, consolidated and arranged on a single screen so the information can be monitored at a glance [25]. LA gives usage and detailed information about an educational portal. It makes use of “indicators” which are specific calculators with associated visualizations [14]. “Wiki usage in last 10 days” is an example of an indicator. But in existing systems these indicators have been traditionally designed and fixed by developers keeping certain aspects in consideration. It is believed that this approach restricts the flexibility of the Learning Analytics tool.

In this master thesis, it is planned to eliminate the inflexibility of indicator definition at system run time. The system is to be made more dynamic and goal-oriented by introducing active participation of the stakeholders in the definition of the indicators. This leads to “personalization” of the system. The system becomes user driven or the user is at the center in the entire scenario. It is the user who is in charge of defining a question, selecting a set of indicators or defining a new one to get result of his question. In a very non-technical sense it would mean that it is planned to develop a system which has some predefined set of indicators but also permits formulation of indicator and question at run time by the user. The user makes selections in terms of data to be fetched and filters to be applied. This user selection is parsed to generate an indicator with the help of a Rule Engine. The indicator generated can be executed to fetch relevant data from the database and display visually to the user.
1.2 Objectives

The overall goals which are strived to be achieved, can be summarized as:

1. The system should help the users in defining their own personalized Learning Analytics question.
2. It should permit to view the list of indicators available.
3. It should permit the user to define new indicators by customization of existing ones.
4. It should support to select different options to visualize the indicator.
5. It should allow the user to add different kinds of data filters to an indicator.

1.3 Research Questions

The objectives ultimately strive to answer the following research questions:

1. How can we make a LA system more dynamic and add elements of Personalization to it?
2. Is such a system useful as well as usable?

1.4 Outline

The rest of the thesis is organized in five chapters. Chapter 2 briefly describes the fundamentals regarding LA systems and the related work in the field of personalized learning analytics. In Chapter 3 the conceptual approach of developing our system has been discussed. Following, Chapter 4 presents the used technologies and the details of the implementation. Chapter 5 deals with user evaluation and the results obtained from it. Chapter 6 concludes the thesis with justifying the fulfilment of the thesis objectives and also throws light on the nature of future work which can be leveraged on the basis of this work.
2 Related Work

“If you steal from one author it’s plagiarism; if you steal from many it’s research.”
–Wilson Mizner

This chapter first deals with the basics of learning analytics. It proceeds to discussion of open learning analytics. Some of the existing LA systems which are being used to various degree of success has been described in detail. A discussion related to inadequacy of existing systems and why these do not fit into the context of open learning analytics has been made. Existing LA systems generally do not offer an optimum level of flexibility and independence to the user. The user is forced to use the default statistical counters which are provided by the system. Normally these are pre-programmed. Inclusion of new statistical features would require additional programming which would require constant maintenance of a development team as well as huge cost investments. Hence in this chapter the existing systems have been studied to formulate a solution which would provide flexibility to the user. That would mean personalization of the LA system.

2.1 Learning Analytics (LA)

Attention is a core concern in learning: as learning resources become available in more and more abundant ways, attention becomes the scarce factor, both on the side of learners as well as on the side of teachers [30]. LA is quite similar to other popular domains like Big Data [1], e-science [56, 33], web analytics [21], Educational Data Mining (EDM) [54]. These fields invariably rely on collection of large quantity of data and detect patterns to come to any kind of conclusion. The data analysed can also be visualized so that teacher/students can understand them. But visualization of learning activities is a challenge in itself [25].

To apply Learning Analytics at a broader scale a good infrastructure with use of standard tools and protocols is highly necessary [26]. For learners and teachers it can be useful to have visual overview of learning activities [25]. A LA system using the above concepts have already been deployed at Purdue University for increasing student success. The results have been quite good [2]. But the major problems associated with LA still continue to exist like [25]:

1. Lack of clarity of what exactly should be measured to understand the learning process.
2. Data privacy issues [22].

2.1.1 LA Reference Model

LA has been defined differently by different researchers. The most widely used definition for LA which was adopted by the first international conference on LA and knowledge (LAK11) is “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” [60]. In simple terms, LA focuses on deriving useful insights from educational data and thereby improving the efficiency of both teaching as well as learning process. A systematic overview on LA and its key components has been devised with the help of a reference model [15]. The model is shown in Figure 2.1.

![LA Reference Model](image)

**Figure 2.1: LA reference model [15]**

The LA reference model has four major dimensions. They are:

1. **What?** What kind of data do the system gather, manage and use for analysis? It also deals with environments and context from which data comes or learning occurs.
2. **Who?** This dimension focuses on different stakeholders who can be benefited from the application of LA including students, teachers, educational institutions, researchers...
and system designers. These varied group of stakeholders generally have different perspectives, goals and expectations from LA.

3. **Why?** This dimension deals with the objectives of LA. It helps answers why the data collected is being analysed. The objectives generally differ with respect to the stakeholders involved. Some of the objectives of LA include monitoring, analysis, prediction, intervention, tutoring or mentoring, assessment, feedback, adaptation, personalization, recommendation, awareness and reflection.

4. **How?** This dimension deals with the methods generally used to detect interesting patterns in the educational data collected. Some of the methods include statistics, information visualization (IV), data mining (DM) and social network analysis (SNA).

### 2.1.2 Process of LA

Firstly the educational usage data is collected from a LMS, Virtual Learning Environment (VLE) or a Personal Learning Environment (PLE). This data is stored in the database and subjected to analysis later. In the step of data mining, mining methods are applied to pre-processed data. The results are graphically visualized based on which teachers can evaluate the students’ performance and improve their teaching methods or vice-versa [14]. The process of learning analytics is shown in Figure 2.2.

![Figure 2.2: Learning Analysis Process [14]](image-url)
2.1.3 Open Learning Analytics

Open learning analytics is the emerging research dimension of LA. The concept was first introduced in 2011 by a group of leading thinkers on LA in an initial vision paper published by the Society for Learning Analytics Research (SoLAR) [60]. The basic idea of open learning analytics is to build a conceptual and technical framework around which different stakeholders could network and share their best practices. This needed the usage of open source software, open standards and open APIs to handle all complexities related to interoperability. But the concept of open learning analytics is still not well defined and several important questions remain unanswered which include but not limited to [16]:

1. How should “open” be interpreted in relation to learning analytics?
2. How can open learning analytics be leveraged to foster personalized, networked, and lifelong learning?
3. What are the challenges in open learning analytics in addition to interoperability and privacy?
4. What are the components of an open learning analytics ecosystem?
5. What are concrete user and system scenarios that an open learning analytics platform should support?
6. What are the requirements for an effective open learning analytics platform?
7. What are the technical details (i.e. architecture and components) of an open learning analytics platform?

The term “openness” has received a great deal of attention and generally interpreted with respect to [16]:

1. **Open learning** By providing understanding into how learners learn in open and networked learning environments and how learners, educators, institutions, and researchers can best support this process [15].
2. **Open practice** Gives effect to a participatory culture of creating, sharing, and cooperation.
3. **Open Architectures** processes, modules, algorithms, tools, techniques, and methods that gives the freedom to use, customize, improve, and redistribute the entities above without constraint.
4. **Open access** It allows different stakeholders to access learning analytics platforms without any entry requirements in order to promote self-management and creativity.
5. **Open participation** It permits to participate in the LA process by engaging different stakeholders in the LA exercise.
6. **Open standards** It helps “to reduce market fragmentation and increase the number of viable products” [20]. Open standards and specifications can help to realize the benefits of better interoperability [19].
7. **Open assessment** It helps lifelong learners gain recognition of their learning. Open assessment is an agile way of assessment where anyone, any time, anywhere, can participate towards the assessment goal. It is an ongoing process across time, locations, and devices where everyone can be assessor and assessee [15].
2.1. Learning Analytics (LA)

8. Open learner modelling. It is based on user interfaces that enable reflection, planning, attention, and forgetting and that can be accessed by learners to control, edit, update, and manage their models \[38\]. This is important to build trust and improve transparency of the LA practice.

The concept of open learning analytics covers all the aspects of “openness” enumerated here. With respect to the LA Reference model, it refers to an ongoing analytics process that encompasses diversity at all four dimensions of the model \[16\]:

1. **What?** It accommodates the considerable variety in learning data, environments, and contexts. This includes data coming from traditional education settings (e.g. LMS) and from more open-ended and less formal learning settings (e.g. PLE, MOOC, social web).

2. **Who?** It serves different stakeholders with very diverse interests and needs.

3. **Why?** It meets different objectives according to the particular point of view of the different stakeholders.

4. **How?** It leverages a plethora of statistical, visual, and computational tools, methods, and methodologies to manage large datasets and process them into indicators and metrics which can be used to understand and optimize learning and the environments in which it occurs.

2.1.4 Open Learning Analytics Platform (OLAP)

To improve the learning and teaching experience in today’s networked and increasingly complex learning environments, LA needs to be shifted from closed tools to LA ecosystems where everyone can contribute and benefit. In an open learning analytics ecosystem stakeholders from different universities with diverse needs and objectives use the system to analyse their datasets. The main users of the project are:

1. **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.

2. **Teachers** Uses the OLAP platform as a complement of the Learning Management Systems. Utilizes data visualization of the OLAP to personalize dashboards with information about student performance, comparison with cohorts, monitoring of activity.

3. **Developers/Researchers** Can create new Analytics Methods and 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.
Chapter 2. Related Work

The technical architecture of such a system is shown in Figure 2.3. Such a system involves many challenges including but not limited to [16]:

1. **Data Aggregation and Integration** A key requirement is to collect data from multiple heterogeneous sources varying greatly in data formats, and consolidate them into a working set which the system can explore and interpret.

2. **Modularity** The system architecture should be modular in nature so that it can be developed or extended by different collaborators.

3. **Flexibility and Extensibility** An open learning analytics platform should be fully flexible and extensible by enabling a smooth plug-in of new modules, methods, and data after the platform has been deployed.

4. **Transparency** Data and interpretations in LA might be used in other than the intended ways. For instance, learners might fear that personal data will not be used for constructive feedback but for monitoring and grading. This could lead to the unintended effect that learners are not motivated to use LA tools and participate in analytics-based TEL scenarios. Transparency is vital to drive forward the acceptance of LA. It provides an explicit definition of means how to achieve legitimacy in the process of learning analytics. It should be applied across the complete process, without exceptions. This means that at all times, there should be easily accessible and detailed documentation of how the data is collected, who has access to the data, which analytics methods are applied to the data, how long is the data valid and available, the purposes for which the data will be used, under which conditions, and which measures are undertaken to preserve and protect the identity of the learner [15, 53, 16]. Further, it is important to increase institutional transparency by clearly demonstrating the changes and the added-value that LA can help to achieve [24].
5. **Personalization** It is important in order to fulfil the needs and goals of multiple stakeholders the system engages end users in a continuous inquiry-based LA process, by supporting them in setting goals, posing questions, interacting with the platform, and self-defining the indicators that help them achieve their goals.

The above aspect of personalization which allows different stakeholders to define their own questions and indicators is handled by the “Indicator Engine” component of the open learning analytics system. The various scenarios handled by the component are:

1. **Teacher Scenario** A teacher utilizes an external LMS system to administer courses. The teacher has access to a personalized dashboard of the OLAP 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.

2. **Student Scenario** Uses the 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 system. 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.

3. **Developer Scenario** Developers and Researchers can develop and register third party application data collection to the OLAP. The applications can upload the data to the OLAP 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 analyse the data and these will be made available to all users. The developer or researcher can also create indicators or use existing ones.

The user selects a goal and enters a question in the question / indicator editor. The system has a component which analyses the question and suggests similar questions. 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 system makes suggestions of instances of the indicator. The user can select those instances or create a new instance with different method and visualization techniques. This is the main focus of this thesis work. Before conceiving a conceptual approach to actualize the system, we start by studying the existing LA systems and understand why they will not be able to cater the needs of open learning analytics.
2.2 Existing LA Systems

This section explains some of the widely used LA systems. They generally help to extract information from online student interaction log. The data is extracted, parsed, analysed and finally presented to the stakeholders for feedback and information purposes.

2.2.1 Academic Analytics Tool (AAT)

In online learning environments, the teachers generally do not get enough feedback about how the students are learning. But these learning environments store huge amount of log information. AAT is a tool designed to exploit the huge amount of log available and provide feedback. The tool can provide valuable information about students’ learning process allowing the identification of difficult or inappropriate learning material and contribute to the design of improved student support activities and resources[31].

The tool allows the users to access and analyse student behaviour data in learning systems. It gives information about student interaction and stores the result in HTML/Comma Separated Values (CSV) files. It is primarily meant to satisfy the needs of course instructors. The course instructors also need to know which course they want to investigate using the tool.

AAT allows the users to specify what kind of data they are interested to know and the kind of analysis they want to perform on it. It can also analyse multiple datasets belonging to different courses. It is quite adaptable to most modern LMSs like Moodle [18], Sakai [57], Desire2Learn [62]. It is expandable with respect to addition of additional analytics modules.

The architecture of the tool is shown in Figure 2.4. It is based on the architecture of De-Les [32], a tool for identifying different learning styles from online student behaviour. The AAT tool accepts user data as input from database, extracts and analyses it ans stores the result in HTML formats. Learning objects help to distinguish data to which domain it belongs to like forum, quiz or learning resources. It makes use of “Patterns” which are based on types of learning objects and specify what data the user is interested to investigate. A pattern is a query that extracts specific data. For example: The average amount of time each student spent on quizzes, the number of times a discussion forum has been visited by the students. “Templates” make the tool compatible with different learning systems. Patterns help in specifying what data should be extracted while templates specify where the data resides in the database. A profile is used for extracting and analysing a particular information. In the profile set-up, the AAT tool guides the user to select a template, specify a pattern and connect to a database.

2.2.2 LeMo

LeMo is an open source application for learning analytics which collects data about learner’s activities from different platforms. Its main is to monitor the learning process. It strives to serves the needs of teachers or course instructors in particular. A survey was conducted among the course instructors and based on the results the system is built with a set of pre-defined indicators. These indicators help to analyse the data collected from learning management
2.2. Existing LA Systems

systems. The system architecture is shown in Figure 2.5. It is basically a 3-tier application. The first tier contains a data model as well as functionality for data management, which includes connecting to different learning platforms. Data analysis takes place in the second tier while the results of the analysis are presented in the third tier.

![Figure 2.4: Architecture of Academic Analytics Tool [31]](image)

**Figure 2.4: Architecture of Academic Analytics Tool [31]**

2.2.3 Explanatory Learning Analytics Toolkit (eLAT)

eLAT is a LA system developed in RWTH [14]. It has been implemented to explore data patterns in a VLE. The data model generated can fit in any VLE including Moodle. It uses asp.net 4.0 and WCF services for its functioning. The exploratory Learning Analytics Tool (eLAT) serves teachers to explore and correlate content usage, user properties, user behaviour, as well as assessment results [14]. eLAT was designed to demonstrate the usage and properties of a learning environment. Hence it makes use of Indicators: which are specific calculators with

![Figure 2.5: Architecture of LeMo [45]](image)

**Figure 2.5: Architecture of LeMo [45]**
corresponding visualization [14]. These indicators have been categorized and some additional parameters can be passed or removed from them. The system is always initialized with a set of predefined indicators which are either in an activated or deactivated state depending on the context of usage. Additionally it focuses mainly for the benefit of teachers.

Figure 2.6 shows the indicator execution process in eLAT. Indicators are UI components that fetch eLAT caters to specific user requirements that has been collected through literature survey, survey conducted among teachers, system administrators at RWTH. The main goals satisfied by the tool are [14]:

1. **Usability** Prepare an understandable user interface (UI), appropriate methods for data visualization, and guide the user through the analytics process.

2. **Usefulness** Provide relevant, meaningful indicators that help teachers to gain insight in the learning behaviour of their students and support them in reflecting on their teaching.

3. **Interoperability** Ensure compatibility for any kind of VLE by allowing for integration of different data sources.

4. **Extensibility** Allow for incremental extension of analytics functionality after the system has been deployed without rewriting code.

5. **Re-usability** Target for a building-block approach to make sure that re-using simpler ones can implement more complex functions.

6. **Real-time operation** Make sure that the tool-kit can return answers within microseconds to allow for an exploratory user experience.

7. **Data Privacy** Preserve confidential user information and protect the identities of the users at all times.

![Figure 2.6: Indicator execution process [14]](image-url)
2.2.4 Workbench

An Analytics workbench was conceptualized and implemented [29] keeping the following points in consideration:

1. The tool provided an explicit representation of analysis work flows. For this they used a graphical language based on pipes and filters design pattern. Every step of the analysis is visualized as an graphical element and the data is made to flow to another component by linking the first component to the other component. The concept is very similar to Yahoo Pipes. Refer to Figure 2.7 for a graphical illustration of the working of the Analytics workbench.

2. The tool should support easy integration of additional analysis functionalities.

The workbench supports a wide spectrum of analyses and can be easily applied to a wide range of data. This includes data from arbitrary learning environments, since lots of standard data formats are supported [29]. But their main area of focus is integration of this workbench in any arbitrary learning environment by mapping the log data to a format on which workbench can operate. The user cannot define any additional analytics in the System.

![Figure 2.7: Analytics Workbench in Action [29]](image)

2.2.5 Portable Learning Analytics

A great amount of work has been done to make a Learning Analytics system portable. In general Learning Analytics solutions can be broadly classified into 3 categories [65]:

1. **Tightly Coupled Learning Analytics Systems** These are developed for a specific platform using platform specific APIs and do not work with other learning platforms.
2. **Pluggable Learning Analytics Solutions** These are integrated with a learning platform using the plug-in service. But still a wide number of plug-in implementations are required for different platforms.

3. **Standalone Learning Analytics** These are of 2 sub-kinds
   a) General web analytics (Google Analytics) are quite independent in nature. But they are too general i.e they neglect the underlying domain specific information and suffer from heavy privacy issues.
   b) Solutions developed on the Contextualized Attention Metadata (CAM) specifications too suffer from different vocabulary specifications.

Learning Analytics portals are heavily platform dependent. Changing the platform is quite cumbersome as the developers are confronted with schema changes and vocabulary changes. An approach for achieving platform independence for Learning Analytics solutions was to implement them using widgets which make use of OpenSocial API\(^1\) to fetch data for analysis and visualization \([65]\). Such a solution has three aspects:
   1. The Presentation tier consists of the Learning platform’s User Interface (UI). A learning dashboard is constructed from widgets\(^2\).
   2. The logic layer provides access to the activity Streams and open social APIs which communicate with presentation and logic tier.
   3. Data persistence tier stores the activity and social data.

### 2.3 Limitations

The above examined LA systems commonly used in universities have a wide range of features. But they are quite restricted as they cater to the needs of a specific group of teachers, researchers and students. Also the indicators available in these systems are statically defined and no run time additions or customizations are allowed. These systems cannot be used in the context of “Open Learning Analytics” because the goal is to develop a platform which is freely available and practically has the least number of restrictions. The Open LA gives special emphasis to “open access” and “open participation” where by different groups of researchers, teachers and students from different Universities can make use of a single platform to analyse their data and derive useful insights. Therefore the platform has to deal with diverse data formats and must cater to diverse needs of a very diverse group of users. In such a scenario, developing a LA system which has every possible kind of indicators to satisfy each and every user is not practically feasible. Hence, a method to dynamically generate indicators during the system runtime is unavoidable. The user should be able to define an indicator from scratch or tailor an existing one to meet his/her needs. This would satisfy the constraints and context of Open Learning Analytics.

The work discussed above has been extended in a sense that now the users are allowed to define custom indicators. Personalization of the LA system is the core objective of the thesis

---

1. **OpenSocial** is a public specification that defines a component hosting environment (container) and a set of common application programming interfaces (APIs) for web-based applications \([11]\).
2. **Widget** : A software widget is a relatively simple and easy-to-use software application or component made for one or more different software platforms \([9]\).
work.

2.4 Summary

Open learning analytics strives to provide an LA ecosystem 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. It has to deal with data coming from different sources which vary greatly in structure. It also has to satisfy different objectives of a diverse group of users. The existing LA systems fail to satisfy this constraint. Hence it is proposed to build a tool which will remove this inflexibility. The existing works done in the field of personalized LA has been examined and in the proceeding chapter, the conceptual approach to frame Questions and Indicators during run-time is discussed.
3 Conceptual Approach

“Thinking fragments reality - it cuts it up into conceptual bits and pieces.”

–Eckhart Tolle

The goal is to build a LA system which would allow the users to define their data selection and visualization during the run time instead of having them to be coded statically. To develop such a system, an abstract understanding of how such a system works is required. The process of data selection and other relevant parameter specification needs to be analysed. This analysis culminates in the conceptual approach on which the system will function. After the identification of research questions, going through various related works and literature, the working approach of the system has been conceptualized.

3.1 Scope and Requirements

Existing LA systems like eLAT fulfil all the requirements of a full fledged LA system. It allows to tap information about the teaching and learning process. But normally these systems making use of LA have a set of predefined calculators with associated visualizations known as “indicators”. The user is forced to use these indicators to know some vital statistical information. In this thesis work, the aim is to extend this scope where the system permits run-time addition of new indicators or customization of existing ones. Conceptually, the user is given liberty to frame his own indicators which he likes the system to evaluate and interpret some results for him. These indicators on execution will fetch relevant data and present the information visually to the user.

The overall requirements of the system are:

1. It should permit addition of new indicators.
2. It should allow customization of existing ones.
3. Successful execution of indicators should be followed by suitable graphical visualization.

Software Requirements deals with establishing the needs of stakeholders that are to be solved by software. The IEEE Standard Glossary of Software Engineering Technology defines a software requirement as [35]:

19
Chapter 3. Conceptual Approach

1. A condition or capability needed by a user to solve a problem or achieve an objective.
2. A condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed document.
3. A documented representation of a condition or capability as in 1 or 2.

A functional requirement defines a function of a system and its components. A function is described as a set of inputs, the behaviour, and outputs. Functional requirements may be calculations, technical details, data manipulation and processing and other specific functionality that define what a system is supposed to accomplish. Behavioural requirements describing all the cases where the system uses the functional requirements are captured in use cases. Functional requirements are supported by non-functional requirements, which impose constraints on the design or implementation [27]. The detailed functional requirements of the system are:

1. The system should allow the user to define a new indicator.
2. It should allow the user to select multiple data sources for an indicator.
3. It should allow the user to add filters to an indicator to refine the data fetched.
4. It should allow the user to specify graphing options for an indicator.
5. It should allow the user to preview the indicator before saving it.
6. It should allow the user to add multiple indicators to one question.
7. It should permit the user to get an overview of the indicators already defined.
8. It should allow the user to view or edit the indicators already finalized.
9. It should allow the user to delete any of the finalized indicators from the question set.
10. The user should be able to search for a specific indicator in the system, view its property and load it into the editor to use it as a template.
11. The user should be able to execute questions and indicators already existing in the system.
12. The user should be able to view the property of the indicators already existing in the system.

3.2 Abstract Architecture

The system’s abstract architecture is shown in Figure 3.1. The working concept of the system is quite straightforward:

1. The user interacts with the “Indicator Editor” which is the UI of the system. There the user can specify what kind of data he/she is interested to visualize by making selections.
2. These selections are passed to the “Indicator Engine” where they are parsed.
3. The parsing is governed by pre-defined rules present in the “Rule Engine”.
4. After the above processing step, the indicator is ready to be executed in the system i.e fetch relevant data from the local data source and present it visually to the user.
### 3.3 Working Scenario

The overall flow of the indicator definition process has been described below with relevant graphical illustrations.

#### 3.3.1 Data Source

The data is fetched from a data source which is populated and maintained by another team of developers and it is not a part of the core application. The organization of the data in the data source can be summarized with the help of an ER\(^1\) diagram. Refer to Figure 3.2.

The ER diagram shows how the system stores the on-line activities of a user. A single user can have multiple events. One event is characterized by a unique time stamp and session values. “Action” indicates a particular Learning Management System (LMS) like L2P to which the data belongs. “Platform” indicates using what device the data was accesses e.g., web or mobile device. This single event can interact with multiple entities like wiki, discussion forum, learning material etc. Each of these entities belong to a separate “category”.

To extract data about events, the user defining an indicator must specify the source i.e the LMS portal whose data he/she is interested in visualizing. The platform must also be specified which helps in filtering events for a particular LMS initiated from the appropriate device like web or mobile. Action indicates if an event was an “Update”, “START” or “END” event. The category to which an event belongs to also needs to be specified. For example if a user specifies:

1. Source: L2P
2. Platform: WEB
3. Action: UPDATE

---

\(^1\) **ER Diagram**: Entity Relationship Diagram. It is a data model for describing the data organization in an abstract way.
4. Category: WIKI
   The system will fetch all update events done to wiki from web to L2P. This selection can be further refined by specifying values for user, session, time-stamp or specific key values.

3.3.2 System and its functionalities

The core application is a web application. Refer to Figure 3.3 for conceptual diagram of the deployed web application. On successful login displays:
   1. An editor to define new question and indicators.
   2. Displays the available list of questions and indicators.
   3. A display area where the user can browse and execute questions.
3.3.3 Overview of User and System Interaction

The System’s in-built Editor is used for defining new indicators. It allows the user to select an entity from a drop-down menu and after that the user selects specific properties of that entity which he likes to know about. The user types a question name in the text box. If the question is “new” then the system allows it. If a matching question does not exist in the database, then such questions are termed as “new”. If the question name matches with some of the questions already existing in the database then the system does not allow it. The user is allowed to choose one or more from the existing indicators as a template or is free to use the in-built editor for defining a new indicator.

The user can add indicators from scratch or modify exiting ones. Every individual indicator can be previewed before associating it with the question. After final selection of a set of indicators for the question, it can be executed. The user gets a textual summary as well as graphical illustration(visualization) as the final outcome to the question. The different individual indicators can also be combined to create composite ones. After a final selection of the indicators, the question can be executed. The list of successfully executed questions and their associated set of indicators are stored in database such that they can be retrieved for reuse in future. This storage feature will also help in improved recommendation as time progresses.
3.3.4 Indicator Generation

The application makes use of Rule Engine to generate an equivalent indicator from the user selections made in terms of what data to be fetched and filters to be applied. The Rule Engine decides if from the user selections a meaningful indicator can be generated which would fetch any relevant information from the available data.

1. If YES, a corresponding indicator is generated and executed to fetch data from the database. The data is forwarded to the application where it is visually presented to the user.
2. If NO, then the system informs that it cannot generate an indicator with the current user selections.

The system can be used by the teachers for close monitoring of the student behavioural patterns and activities. The system is also accessible to students for an in-depth analysis of how well their study is progressing, they get to know how is their performance in comparison to others etc. A place of “self-reflection” and “self-assessment” for the students! The main focus is to make the LA indicator “dynamic” in nature such that it also allows the user to frame their own indicators with in the domain of available information. To achieve this kind of partial automation help of a Business rules engine is sought. The system keeps track of the newly added indicator and it is available for reuse in future. With this approach the pool of indicators in the system increase continuously. This leads to a plethora of indicators as well as questions available with the system for reuse.

3.4 Use Case Scenarios

The Use Case Model describes the proposed functionality of the new system. A Use Case represents a discrete unit of interaction between a user (human or machine) and the system. A Use Case is a single unit of meaningful work. For example login to system, register with system and create order are all Use Cases. Each Use Case has a description which describes the functionality that will be built in the proposed system. A Use Case may ‘include’ another Use Case’s functionality or ‘extend’ another Use Case with its own behaviour. An Actor is a user of the system. This includes both human users and other computer systems. An Actor uses a Use Case to perform some piece of work which is of value to the business. The set of Use Cases an actor has access to defines their overall role in the system and the scope of their action [67].

3.4.1 General Functionalities Use Case

The proposed system’s functionalities can be broadly classified into four major categories. The user while using the system is able to:

1. Execute existing indicators.
2. View the details of existing questions and indicators.
3. Define a new question with some indicators.
4. Modify an existing indicator and save it as a new one. This interaction of user with the
3.4. Use Case Scenarios

Figure 3.4: Overall Use Case Scenario

The most important goal of this thesis work is to permit run-time addition of new indicators. As the system strives to satisfy varying needs and objectives of a wide range of stakeholders, this feature is of paramount importance. It is this uniqueness of the system which makes it eligible to integrate into an open learning analytics ecosystem.

The Use Case of an user trying to define a new indicator is depicted in Figure 3.5. The basic outline of the scenario is:
Chapter 3. Conceptual Approach

1. First the user enters the Question-Indicator editor and selects the preliminary information like question name, indicator name and the data sources i.e which particular data the user is interested to visualize. The source basically indicates data coming from a particular LMS or a group of LMSs.

2. In the second step, the user can select which category of data of the selected source the user is interested to know. It can range from learning material resources, wiki to discussion forum etc.

3. In the third step, the user must specify how the data fetched can be presented back visually i.e the user can specify various charting options like bar graph, line graph etc.

4. In the final step, the user can preview the indicator to check if it satisfies his/her constraints. The indicator parameters can be changed again if desired. When the user is satisfied the indicator can be saved to the database for future execution.

3.4.3 Filter Addition to Indicators Use Case

![Diagram of Filter Addition to Indicators Use Case]

During the process of indicator definition the user in addition to specifying basic information like indicator name, data source, data category and graph options can also add information to refine the data fetched. These additional pieces of information are called filters. The Use Case scenario for adding a filter to an indicator is captured in Figure 3.6. The basic outline of the scenario is:

1. The user enters the Question-Indicator editor and does the specification of preliminary information. This includes specifying a question name, an indicator name, selection of data sources, selection of data category within selected data sources and graph options.

2. In the second step, the user can add filters to refine the data fetched. The information used for restricting the data fetched can be for a specific user, specific session value, specific time slots or specific data attributes which are known as user filters, session
filters, time filters and attribute filters respectively. The user can add these filters in any combination or in any number.

3. The indicator can then be previewed and can be subjected to further modification which may include change of preliminary settings, inclusion of additional filters or deletion of existing filters.

4. When the user is satisfied the indicator together with all its filters can be saved to database.

3.4.4 Indicator Modification Use Case

The most important Use Case scenario provided by the system is to reuse the existing indicators. This is captured in Figure 3.7. The user can modify existing indicators to create new ones.

The basic outline of the scenario is:

1. The user can search for a specific indicator which is existing in the system and can view its properties. The indicator can be loaded into the Question-Indicator editor for customization.

2. In the second step, the user must provide a new indicator name. The user is also free to alter the data source, data category, add new filters to the indicator, delete existing ones or change the graph options.

3. The indicator can be previewed and changed till the user is satisfied. Finally the new custom indicator is saved to the database.
3.5 Summary

The system’s objective to allow run-time addition of new indicators and customization of existing ones is a highly challenging one. In this chapter, an abstract architecture based on which the system has been implemented was discussed. The conceptual approach of the system’s functioning including the details of data source on which the system runs, the user-system interaction and an overview of indicator generation process was examined. The most important functionalities provided by the system are also modelled using Use Case diagrams.

The next chapter deals with insights from the actual implementation scenario which includes technologies used and also detailed information of various components which work collaboratively to meet the system’s functional requirements.
4 Implementation

“Always code as if the guy who ends up maintaining your code will be a violent psychopath who knows where you live.”

—Martin Golding

In this chapter, the implementation of the system has been described. It first begins with a brief introduction of the technologies used, followed by explanation of the architecture of the system. The various components of the system including their role, working and implementation details has been described. Special emphasis has been given to the Question-Indicator editor with which the user interacts to define an indicator. All its sub-components and functionalities has been dealt with exhaustively.

4.1 Introduction

Indicator Engine is a Spring Framework based Java application. It is designed to help the users define their own set of questions and indicators to query the existing data set. It also provides them with enough flexibility to customize existing questions and Indicators. The application makes extensive use of web technologies including both client as well as server side to fulfil the user requirements. It is an attempt to fulfil the personalization requirement of the LA i.e the system becomes user driven.

4.2 Technologies

The application uses the following technologies for its functioning:

1. Spring Framework The Spring Framework is an application framework and inversion of control container for the Java platform [66]. It is a good replacement to Enterprise JavaBeans (EJB) [46]. It can be used by any Java application but especially used for building web applications on top of JavaEE platform [42].

2. Spring Model-view-controller (MVC) The Spring Framework has its own MVC web application framework [39]. It has many improvements over other frameworks like Struts [34]. It is a request based framework. The framework defines interfaces which handle specific tasks and they are tightly coupled to the Servlet API [42].
3. **Spring Security** Spring Security is a Java EE framework provides authentication and authorization for enterprise application [44, 47].

4. **Hibernate** It is a Object-Relation mapping framework for Java to do object oriented domain model to a relational database [4].

5. **Drools** It is a business rules management system. Contains a set of rules and their associated actions.

6. **Cewolf** It is a Java Framework for generating graphs inside a servlet or a Java server page (JSP).

### 4.3 Architecture

Bass, et al. define architecture as:

“The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them. Architecture is concerned with the public side of interfaces; private details of elements-details having to do solely with internal implementation are not architectural [61].”

The Architecture of the system usually depicts how the system will function. It is a blueprint used during system development and maintenance. Analysis of the architecture will reveal the productivity, usefulness and feasibility of the system. It can also help to identify potential risk early in the software development so that steps can be taken to mitigate them. Implementation typically includes software development, documentation, testing and bug-fixing to successfully develop a software product that meets all the stated objectives and has a high level of user satisfaction. The application with the help of technologies explained above functions on the web. The abstract architecture and its various components is shown in Figure 4.1.

### 4.3.1 Indicator Engine Dispatcher Servlet (IEDS)

**Dispatcher Servlet** is the starting servlet of the application which is provided by Spring MVC. It is responsible for accepting all incoming user requests. Based on the user request which has come, the IEDS queries the URL handler mapping to resolve and accordingly delegate the request to the spring MVC controller for further processing. Listing 4.1 shows a typical Dispatcher servlet configuration in *web.xml*. It mainly contains the name of the servlet and the kinds of web links it will respond to. It also contains the file name where additional configuration parameters can be specified.

**Listing 4.1**: Indicator Engine Dispatcher Servlet XML File Configuration

```xml
1 <servlet>
2   <servlet-name>IndicatorEngine</servlet-name>
3   <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
4   <init-param>
5     <param-name>contextConfigLocation</param-name>
6     <param-value>/WEB-INF/servlet-context.xml</param-value>
7   </init-param>
8 </servlet>
```
4.3. Architecture

**Figure 4.1: Indicator Engine Architecture**

```xml
<param-value>/WEB-INF/mvc_dispatcher_IndicatorEngine-servlet.xml</param-value>
</init-param>
<load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
  <servlet-name>IndicatorEngine</servlet-name>
  <url-pattern>/</url-pattern>
</servlet-mapping>
```

### 4.3.2 URL Handler Mapping

The **URL Handler** mapping contains a set of URL patterns and an associated controller. If a particular user request matches a specific pattern then the associated controller is invoked for
4.3.3 Spring MVC Controller

Controllers are Java classes designed to handle specific user requests. They are designed to handle only a specific type of HTTP request namely either a GET or a POST request. It processes the URL requested and returns the name of the corresponding view which the IEDS must process and return to the client. Listing 4.2 demonstrates how a typical Controller looks like and it also shows the URL pattern and the type of HTTP request it is designed to handle. It responds to http://localhost/indicators_definition and informs the IEDS that it must process a view named “indicator_system/number/new_ui” for the client to view the necessary information.

Listing 4.2: A Typical Controller with in the Spring Environment

```java
@RequestMapping ( value = "/indicators_definition", method = RequestMethod.GET )
public String getNewIndicatorDefinitionHome (Map<String, Object> model) {
    IndicatorPreProcessing indicatorPreProcessor = (IndicatorPreProcessing) appContext.getBean("indicatorPreProcessor");
    model.put("selectNumberParameters", indicatorPreProcessor.initSelectNumberParametersObject());
    return "indicator_system/number/new_ui";
}
```

4.3.4 Models

Models are implemented using plain Java. They are designed to hold all the domain business logic and are made from the following three structures:

- **Domain Objects**: These hold all the necessary business logic which are invoked from the controller based on certain criteria.

- **Data Mappers**: These objects are solely responsible for retrieval and storage of requisite data from the Data source using Hibernate.

- **Services**: These are a kind of higher level objects responsible for interaction between the Domain objects and the Data Mappers.

4.3.5 Views

View instances are responsible for the presentational logic. Each View acquires data from the Model Layer and then, based on the received information, chooses a template and sets values which is finally presented to the client. Views are implemented using JSP.
4.3. Architecture

Listing 4.3: A Typical View File

```java
<%@ page import="com.indicator_engine.datamodel.UserProfile" %>
<%@ page contentType="text/html; charset=UTF-8" language="java" %>
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form" %>
<%
if ((session.getAttribute("loggedIn") == null) || (session.getAttribute("loggedIn") == "
response.sendRedirect("/login");
if ((session.getAttribute("loggedIn") != null) && (session.getAttribute("userName") != null) && (session.getAttribute("activationStatus") == "false"))
response.sendRedirect("/activate");
else{
UserProfile up = (UserProfile) request.getAttribute("UserProfile");
%
```html
```
<html>
<head>
<title>User Profile</title>
</head>
<body>
<title>User Profile</title>

4.3.6 View Resolver

*View Resolver* is invoked by the IEDS to build the absolute path of the view file whose name is returned by the controller after it finishes processing of a particular user request. The following listing shows how a typical View Resolver configuration looks like. Here it instructs the IEDS to append a specific suffix and a prefix to the view name which it has received from the controller.

Listing 4.4: A Typical View Resolver Configuration

```xml
<bean id="viewResolver" class="org.springframework.web.servlet.view.InternalResourceViewResolver">
    <property name="prefix">/pages/</property>
    <property name="suffix">.jsp</property>
</bean>
```

4.3.7 CEWOLF Graphing Servlet

*Chart Enabling Web Object Framework* (CEWOLF) servlet is a special dedicated servlet designed to handle all graphing requests coming from the client. It is not being used currently but
will be used in the future when the need for complicated graphing scenarios and the increased frequency of graphing requests will arise.

## 4.3.8 Accessory Beans Running in Application Context

There are some dedicated beans running in the application context which can be invoked by any method with in the IEDS context to do a specific task for example sending a mail to the user or access data from the data source. Some of them have been listed below:

1. **Hibernate** Responsible for accessing the data source using Hibernate Object Relation Mapping (ORM) framework. It is called from all methods for their data storage or retrieval needs.
2. **Password Encoder** Has the necessary methods to encrypt information to be stored in the database or decrypt specific information retrieved from the database.
3. **Mail** Contains methods to send mail to user email id with or without attachments. It has been designed so that the users after generating report from the system can forward a copy of the same to a specific mail address.
4. **Drools** Responsible for generation of Hibernate Query Language (HQL) from the parameters selected by the user in the Indicator definition process.
5. **Threading** Generally used to schedule some asynchronous tasks and all stuff related to threading.
6. **Learning Context DB Access** Contains methods to access and manipulate data directly from the data model of the Context Learning Project.
7. **User Beans** Contains the models which is used by Hibernate to fetch user data from the database.

## 4.4 Data Models

Indicator Engine uses certain database structures for its proper functioning. These have been broadly classified into three categories:

### 4.4.1 System User ER Model

This model is used by the system to organise the information of all users who access it. The login information is stored in an entity called “User Credentials”. All the accessory information about an user is stored in “User Profile”. Every User Credential has a unique User profile. There is a one to one correspondence here. But a user can have several roles (roles denote level of access), hence a single User Credentials can be associated with multiple “Security Role” entities. Figure 4.2 shows this ER model.

1. **User Credentials** User Credentials has the following attributes.
4.4. Data Models

Figure 4.2: System User ER Model

a) **UID**  It stores system generated unique identification number for every user.
b) **User Name**  The name of the user which is used for login purposes is stored here.
c) **Password**  This field stores the secret key used for login purposes.
d) **Activation Status**  It is a boolean field which denotes if the user account is in an activated/deactivated state.
e) **One Time Password**  It stores a unique system generated password which is used for verifying the user during his first login attempt.

2. **User Profile**  User Profile contains attributes required to maintain a full-fledged user profile like date of birth, address etc.

3. **Security Role**  Security Role has one attribute named “Role Name” which contains the roles performed by the user in the system. Roles can be like `ROLEUSER` or `ROLEADMIN`. 
4.4.2 Question & Indicator ER Model

This model is used by the system to store the user defined Questions and Indicators. Figure 4.3 shows this ER model. Every Question has its own unique property collection and so does every Indicator. There is one to one correspondence here. But there is a many to many correspondence between a Question and an Indicator as shown in the ER model. The various attributes in this model are:

1. **Question** Question Entity has the following attributes.
   
   a) **Name** It stores the name of the Question defined by the user.

   b) **Number of Indicators** It stores the numerical value denoting how many indicators are associated with this Question.

2. **Question Property** Question Property Entity has the following attributes.

   a) **User Name** It stores the name of user who created the Question.

   b) **Last Execution Time** It stores time when the Indicator was last executed.

   c) **Total Executions** It stores a numerical value indicating how many times the Question was executed.

3. **Indicator** Indicator Entity has the following attributes.

   a) **Name** It stores the name of the Indicator defined by the user.

   b) **Short Name** It stores a short name for the Indicator. This field is reserved for future use possibly generated by the system to be used for identification purposes.

   c) **HQL** It stores the Hibernate Query for the Indicator which is generated by Drools and parser methods. When an Indicator is executed in the system, the query is retrieved from this field and executed to fetch data from database.

4. **Indicator Property** Indicator Property Entity has the following attributes.

   a) **User Name** It stores the name of user who created the Indicator.

   b) **Last Execution Time** It stores time when the Indicator was last executed.

   c) **Total Executions** It stores a numerical value indicating how many times the Indicator was executed.

   d) **Chart Engine** It stores the name of the graphical library to be used for graph generation while execution of the Indicator.

   e) **Chart Type** The kind of graph to be used for a particular Indicator is stored here.

   f) **is Composite** It is a boolean field which denotes if the Indicator is built from scratch(false) or built from the aggregation (true) of two or more indicators.

   g) **JSON Data** It stores all the user selections made by the user in the Question-Indicator Editor in the form of a JSON string. This field is used when an
4.5. Activity Scenario: Interaction of Different Components

Indicator is being reused as a Template.

4.4.3 Data Source ER Model

This model essentially represents how the data collected from educational portal is collected and stored on which analytics is performed. The details are given in Appendix. Refer to Figure 3.2 in the Appendix for details of this model.

4.5 Activity Scenario: Interaction of Different Components

The sequence of interaction of various components in the system is essential to know. Figure 4.4 shows the sequence diagram which depicts interaction between a user and system components. The explanation for the numbering given in the figure is as below:

1. The user requests a particular page or URL. This request is first received by Indicator Engine Dispatcher servlet IEDS.
2. On receiving a client request, it sends a message to URL Handler Mapping. The URL Handler Mapping service examines the requested URL and determines the specific controller name responsible to handle that request.
3. The name of the controller is sent back to IEDS.
4. IEDS invokes the specific controller and also forwards the user parameters (if any). This controller can in turn make use of models, make database connections, do complex processing stuff.
5. Controller may return a JSON value or the name of the view to IEDS.
6. In case of a view name, IEDS again invokes the View Resolver.
7. The View Resolver is responsible to construct the full path name of the view resource from its name. It sends this full path name back to IEDS.
8. IEDS picks this view resource and sends back to the user to mark the completion of one user-system interaction cycle. The cycle is repeated in the same way for each interaction instance. But there is a slight difference in in the interaction sequence when the controller returns a JSON value instead of a view name. In that case the IEDS does not invoke the View Resolver, but directly passes the value to the user thereby ending the interaction sequence.
Figure 4.3: Question & Indicator ER Model
Figure 4.4: Sequence Diagram: User and System Interaction
Chapter 4. Implementation

After the Conceptual approach and the Architecture of the system were finalized then the functionalities of the system were implemented. The different components of the system are:

1. Object Relation Mappings
2. Controllers
3. Parsing and Query Generation
4. User Interface & Client side scripting
5. Miscellaneous components

4.6 Object Relational Mappings

Modern applications are built using two very different technologies: object-oriented programming for business logic; and relational databases for data storage. Object-oriented programming is a key technology for implementing complex systems, providing benefits of re-usability, robustness, and maintainability [55]. Programmers strongly prefer to work with persistent data held in program objects, rather than use SQL directly for data access, even though this means working around the famous “impedance mismatch” between tabular data and object state [50]. Object Modelling is an approach in which the entire set of information is represented as entities called objects. Unified Modelling Language is widely used for the visualization of object models [5]. Objects can be understood in a more simplified way. For example when we enter a classroom it has projector, blackboard, table, chair, light bulbs etc. Each of these corresponds to an Object. Refer to Figure 4.5 for a diagrammatic explanation of Objects.

<table>
<thead>
<tr>
<th>John : Student</th>
</tr>
</thead>
<tbody>
<tr>
<td>- name = 'John'</td>
</tr>
<tr>
<td>- age=23</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Alex : Student</th>
</tr>
</thead>
<tbody>
<tr>
<td>- name = 'Alex'</td>
</tr>
<tr>
<td>- age=27</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Thomas : Student</th>
</tr>
</thead>
<tbody>
<tr>
<td>- name = 'Thomas'</td>
</tr>
<tr>
<td>- age=25</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Mary : Student</th>
</tr>
</thead>
<tbody>
<tr>
<td>- name = 'Mary'</td>
</tr>
<tr>
<td>- age=23</td>
</tr>
</tbody>
</table>

Figure 4.5: Different Objects of the class : “Student”

The Relational Model was developed in 1970’s by E.F. Codd [17]. It is based on the predicate logic and set theory. The basic concept is that all information is described as predicates and truth statements. It describes only things which are true in the real world. A relational database is a collection of 2D tables [50].

Object to Relational (O/R) mappings bridge the mismatch by converting data to and from a relational database to appropriate objects [6]. With O/R mapping systems like Hibernate [4],
a programmer thinks in terms of entities and their relationships. The system takes over all the complexities of relationships, generation of necessary SQL at run-time. So the business logic programming can be done using Java itself. In the development agenda Hibernate has been extensively used as the ORM framework to handle all database access. The different data models used by the system have been explained in the earlier chapters. Now an example of how this Object-Relational mapping is done is examined. For example - In the system every Indicator has its own set of properties. There is a 1:1 relationship between an Indicator and its property. The corresponding Hibernate annotation is shown in Listing 4.5 and Listing 4.6.

**Listing 4.5: Indicator Hibernate Mapping**

```java
@javax.persistence.Entity
@javax.persistence.Table ( name = "gla_Indicator" )
public class GLAIndicator implements java.io.Serializable {

    @javax.persistence.Id
    @javax.persistence.Column ( name = "indicator_id" , unique = true , nullable = false )
    @javax.persistence.GeneratedValue ( strategy = javax.persistence.GenerationType.IDENTITY )
    @javax.persistenceExpose
    private long id;

    ..... 

    @javax.persistence.OneToOne ( fetch = FetchType.LAZY , cascade = CascadeType.ALL , mappedBy = "glaIndicator" , orphanRemoval = true )
    @javax.persistenceExpose
    private GLAIndicatorProps glaIndicatorProps;
}
```

**Listing 4.6: Indicator Property Hibernate Mapping**

```java
@javax.persistence.Entity
@javax.persistence.Table ( name = "gla_IndicatorProps" )
public class GLAIndicatorProps implements java.io.Serializable {

    @javax.persistence.Id
    @javax.persistence.Column ( name = "props_id" , unique = true , nullable = false )
    @javax.persistenceGeneratedValue ( strategy = javax.persistence.GenerationType.IDENTITY )
    @javax.persistenceExpose
    private long id;

    ..... 

    @javax.persistence.OneToOne ( fetch = FetchType.LAZY )
    @javax.persistenceJoinColumn ( name = "indicator_id" , nullable = false )
    @javax.persistenceExpose
    private GLAIndicator glaIndicator;
}
```

4.7 Controllers

Controllers are Java classes as explained in the previous chapter. They are designed to handle client requests. A controller’s task is very specific i.e it handles only one simple HTTP request and delivers a corresponding view to the user. Controllers in our system have been broadly classified into two major categories:
1. **JSON based Controllers**  These controllers are basically server end points with which the client side scripts interact to retrieve specific information from the server. For example: On the User Interface when the user types a new name for an Indicator, the system must adhere to some specific constraints like minimum number of characters in the name, no duplicate names etc. In order to achieve this functionality, the client side scripts send a HTTP GET/POST request to these controllers with or without additional parameters. These controllers on receiving the request do the necessary processing and finally return appropriate response in JSON format. The client side scripts based on the received response proceed further so that the system performs in an consistent and error-free manner. The definition format of such controllers is shown in Listing 4.7.

2. **View based Controllers**  These controllers are generally invoked in the last step of a page’s functionality. They may or may not validate the data of the page, do some additional processing and return to the user a new page to interact with. The definition format of such controllers is shown in Listing 4.8.

### Listing 4.7: JSON based Controllers
```java
@RequestMapping (value = "/fetchExistingQuestionsData.web", method = RequestMethod.GET, produces = "application/json")
public @ResponseBody String fetchQuestionData (HttpServletRequest request) throws IOException {
    .......
    .......
    }
```

### Listing 4.8: View based Controllers
```java
@RequestMapping (value = "/", "/welcome**", method = RequestMethod.GET)
public ModelAndView LoginController () {
    ModelAndView model = new ModelAndView ()
    model.setViewName("app/welcome");
    return model;
}
```

### 4.8 Parsing & Query Generation

**Indicators** are specific calculators which essentially retrieve relevant data from the database and finally yield a quantitative result. So technically every indicator has an associated hibernate query to fetch relevant data. The hibernate query is generated by the application making use of Drools and some Java based parser methods. The detailed procedure and the modules participating in this process and their implementation specifics has been outlined below:
4.8. Parsing & Query Generation

4.8.1 Data Source and Filter Specifications

The Specification of the data source i.e from where the data is to be fetched for analysis like Source, Action, Platform etc. (refer to ER Model) and also the relevant filters is specified by the end user using the User Interface. The user selects various options from the UI and at the same time applies different filtering criteria to the data. Generally Filters include but are not limited to User Filters, Time Filters, Entity Filters and Session Filters.

Filters help in refinement of data and they are of the following types:

- **User Filters**  User Filters help in refining the data fetched for a particular user.
- **Session Filters**  Session Filters help in limiting the data fetched for a unique session id.
- **Time-Stamp Filters**  These filters help in fetching relevant events within a given time range.
- **Attribute Filter**  These help to search event data for specific key,values.

At the end of user selection all these data is collected by the system and sent to the server by making use of Asynchronous JavaScript and XML (AJAX). Figure 4.6 shows the interface used by the end user to edit Questions and Indicators.

![Figure 4.6: Question-Indicator Editor Interface](image)

4.8.2 Parsing and Query Generation with Drools

When the server receives all the relevant information from the user, it attempts to build a relevant query from the data received. The modules involved in this process are:
4.8.2.1 Operation Number Processor

*Operation Number Processor* is a bean running in the application context mainly responsible for initializing the Drools Engine. It creates a unique Drools session in which it inserts the parameters entered by the user together with a relevant rule file. This Rule file contains all possible scenarios for handling the user input. It also inserts other accessory java objects which may be used by Drools for parsing the user data and in generation of the hibernate query. The creation of a Drools session is typically done as indicated in Listing 4.9.

**Listing 4.9: Drools Session Creation**

```java
StatefulKnowledgeSession session = null;
try {
    KnowledgeBuilder builder = KnowledgeBuilderFactory.
        newKnowledgeBuilder();
    builder.add(ResourceFactory.newClassPathResource("com/
        indicator_engine/indicator_system/Number/
        OperationNumberRules.drl"), ResourceType.DRL);
    if (builder.hasErrors()) {
        throw new RuntimeException(builder.getErrors().toString()
            ());
    }
    KnowledgeBase knowledgeBase = KnowledgeBaseFactory.
        newKnowledgeBase();
    knowledgeBase.addKnowledgePackages(builder.
        getKnowledgePackages());
    session = knowledgeBase.newStatefulKnowledgeSession();
    session.insert(entitySpecification);
    session.insert(new ProcessUserFilters());
    session.fireAllRules();
}
```

4.8.2.2 Drools Rule File

The rule file is the heart of all logic and decision making. It is written using mvel expression language. Typically the rules indicate specific conditions which may arise and the appropriate action for those conditions. Every Drools Rule File has three main components:

1. **Header** This contains a phrase in English to serve as the human identifiable name and can also indicate in brief about the condition in which that rule may be triggered.

2. **When** This is where the condition is specified. Basically Drools checks if the condition holds i.e it evaluates the rule.

3. **Then** This part describes the actions which are to be taken by Drools if the conditions specified in the “When” part are satisfied.

Listing 4.11 shows a typical Drools Rule defined in a Rule File. The three components are clearly visible.
4.8.2.3 Parser Methods

Parser methods are Java methods responsible for parsing a set of user input and generating the conditions in the form of a hibernate query. This resultant hibernate query from one parser method is not complete and is therefore not executable in the Database. It must be combined with the output of other parser methods suitably to generate a working hibernate query. There are several parser methods defined in the System to handle different kinds of user data like Time stamp data, user name and email data, session data etc. All these parser methods are defined within a single Java class and an object of the same is inserted into the Drools session by the “Operation Number Processor”. The different methods remain available to the Drools Engine throughout the session and the rule file specifies which parser methods needs to be invoked under which circumstance and sequence. It also specifies how the output of the different parser methods should be combined to give a full-fledged working hibernate query. This hibernate query represents the desired user selection and on execution will fetch the relevant data from the database. Listing 4.10 shows a portion of code from a parser method. This portion of the code can parse the different sources of data selected by the user and output it in the form of an intermediate hibernate query. This can be combined with the output of other methods by Drools to yield the final query.

Listing 4.10: Parser Method written in Java to parse and generate the relevant query for different sources selected by the user.

```java
public String processSource( List<String> sources, String filter) {
    String hibernateQuery = "(";
    int counter = 0;
    for(String source : sources) {
        if(counter == 0) {
            hibernateQuery += "'" + source + "' ";
            counter ++;
            continue;
        }
        if(counter >= 1) {
            hibernateQuery += ",'" + source + ",' ";
            counter ++;
            continue;
        }
    }
    hibernateQuery += ")";
    return hibernateQuery;
}
```
dialect "mvel"

rule "When Only Entity Filtering (Default Rule) is Present"
when
  EntitySpecification ( userSpecifications.isEmpty() == true,
  timeSpecifications.isEmpty() == true,
  sessionSpecifications.isEmpty() == true,
  entityValues.size()== 1,
  entityValues.get(0).geteValues() == "ALL")
  $pFilter : ProcessUserFilters() 
  $userParameter : EntitySpecification ( $source : selectedSource,
  $platform : selectedPlatform,
  $action : selectedAction,
  $major : selectedMajor,
  $minor : selectedMinor,
  $type : selectedType,
  $key : entityValues,
  $pobject : persistenceObject,
  $filter : filteringType,
  $robject : retrievableObjects)
then
  $userParameter.setHql("SELECT " + $robject + " FROM " + $pobject + " WHERE glaEvent.action =" + $action+ "");
  $userParameter.setHql("SELECT " + $robject + " FROM " + $pobject + " WHERE glaEvent.source IN " + $pFilter.processSource($source, $filter) + " AND glaEvent.platform = " + $platform + " AND glaEvent.glaCategory.major = " + $major + " AND glaEvent.glaCategory.minor = " + $minor + " AND glaEvent.glaCategory.type = " + $type + " AND key = " + $key.get(0).getKey()+ "");
4.9 User Interface & Client side Technology

The User Interface has been implemented using traditional JSP, HTML and CSS.

To make the Question definition process easy and intuitive extensive use of client side technologies has been made. Form validation, user notifications as well as user guidance has been incorporated to the client side. JavaScript libraries like jQuery, jQuery UI, DataTables and noty have been used.

1. jQuery: as well as traditional AJAX has been extensively used for validating user input, dynamically retrieving data from the server by connecting to JSON based Controllers.

2. jQuery UI: This has been used to achieve a cleaner User Interface with special effects to make the Question definition process intuitive.

3. DataTables: It is used to display data dynamically retrieved from server in a structured way.

4. noty: It has been extensively used for user notification like warning, success, error, attention etc. Listing 4.12 demonstrates a code sample for achieving the user-end notification.

Listing 4.12: noty based user notification : Implementation

```javascript
if (request.responseText == "exists") {
    noty({
        text: '<strong>Error</strong> <br/> Question name already Existing. Duplicate names not allowed',
        type: 'error'
    });
    document.getElementById("questionNaming").value=null;
}
```

4.10 Miscellaneous Components

Miscellaneous components add extra value and provide extra features in order to enhance the basic functionality provided by the system. The two most of the notable miscellaneous components out of many additional components that has been implemented are:

1. Mail System: Mail System has been designed to send OTP to user email address for verification. It may be extensively used in the future to send weekly status report.
of Question Visualization to the user email address. The system already supports scheduling of jobs.

2. **OTP Validation** OTP validation mechanism has been implemented during the user registration process in order to prevent misuse of the system and prevent spammers from gaining access to the system. It is basically a java module which generates a unique password and sends to the email address of the user. The user needs to enter this password during the first login attempt made. In case of mismatch, the user account is suspended. The Listing 4.13 demonstrates a bit of its implementation.

**Listing 4.13**: One Time Password Feature : Implementation

```java
public class OTP {

    public static String generatePassword() {
        String chars = "abcdefghijklmnopqrstuvwxyz 
                + "ABCDEFGHIJKLMNOPQRSTUVWXYZ 
                + "0123456789";

        final int PW_LENGTH = 20;
        Random rnd = new SecureRandom();
        StringBuilder pass = new StringBuilder();
        for (int i = 0; i < PW_LENGTH; i++)
            pass.append(chars.charAt(rnd.nextInt(chars.length())));
        return pass.toString();
    }
}
```

Question-Indicator Editor is one of the core components of the system. It is used for defining a new Question in the system. Allowing the user to define his own question and indicators is what leads to active participation of the user and personalization of the system. In this chapter, this component has been explained in detail with regards to the functionalities it provides with relevant graphical illustrations.

When the user accesses the system using the URL there is a specific controller defined which has the responsibility of displaying the login page. After verification of user credentials, the user is redirected to the dashboard of the tool-kit. The user has the option:

1. To view all existing Questions and their associated indicators.
2. To execute a specific Question and its associated indicators.
3. Define a new Question.

When the user selects the option to view all existing Questions in the database, the system typically presents a page (Refer to Figure 4.7) where he can search and view the properties in details. The system displays typical Question properties like:

1. Question Identification Number
2. Question Name
3. Last execution time
4. Number of indicators associated with this Question
5. Number of times executed
6. Owner name.

Similarly, for all the indicators associated with this Question, the System displays the following Indicator Properties:

1. Indicator Identification Number
2. Indicator Name
3. Last execution time
4. Number of times executed
5. Visualization Type\(^6\)
6. Visualization Library\(^7\)
7. Hibernate Query
8. Owner name.

Figure 4.8 shows how the properties of a Question are presented to the user.

---

**Figure 4.7: View Existing Questions in the Database**

The user can select the option of executing a Question which is already existing in the database. The system provides the necessary functionality to enable the user to search and select a Question to visualize. The System then executes the Question with all its associated indicators (i.e., fetches relevant data from the database) and displays them graphically. Refer to Figure 4.9 for a typical Question execution.

The user also has the option to define a new Question which will launch the Question-Indicator Editor.

\(^6\) **Visualization Type**: It specifies the type of graph this indicator will produce during runtime like bar chart, pie chart, etc.

\(^7\) **Visualization Library**: It specifies the type of library that will be used for producing the graph during runtime.
4.11 The Question-Indicator Editor (QI Editor)

The QI Editor is the core of the application. It is the place where the personalization of the Learning Analytics system takes place i.e the user gets to the center and controls the system. The QI Editor typically looks like as in Figure 4.10. The QI-Editor has four major components:
4.12 QI Editor Components

All the major and minor components of the QI Editor play a significant role. When the QI Editor is loaded, the Question definition process is started by default. In the definition process, a Question can have multiple indicators. The user can do an individual Indicator visualization as well as visualize the entire Question together with all its Indicators. When doing a full Question visualization, the user has the possibility of combining two or more indicators and making a new Composite Indicator out of them. This new Composite Indicator will be added to the Question together with the previous indicators. Every Composite Indicator will have an independent name and visualization techniques. It will be independent of the indicators from which it was created and continue to function even if the source indicators are deleted. The user can also use an existing Indicator from the database as a template instead of starting from scratch.
4.12.1 Question Information

Figure 4.11 shows the Question Information component. The user can enter an unique name for the Question. The system does not allow duplicate names for Question. For the first Indicator, the user should enter an unique indicator name. The following operations are available in this component:

![Figure 4.11: Question Information component of QI Editor](image)

1. Template Load  It is used for searching and selecting a pre-existing indicator to be used as a template for the currently being defined Indicator. On clicking this the “Template Load” component gets activated which provides the necessary user interface to search, view the properties of an existing Indicator and finally load those properties into the QI Editor interface.

2. New Indicator This is used to start a new Indicator definition process for the current Question. This is normally selected after the current Indicator has been finalized else the settings present for the current Indicator will be lost.

3. Visualize Question Using this operation the user can visualize the current Question and all its defined indicators.

4. Save Question This operation is used to save the current Question with all its indicators to database.

4.12.2 Question Summary

Figure 4.12 shows the Question Summary component. This component is to be used only after the finalization of at least one indicator. It displays a comprehensive summary for the Question. It displays all the indicators which have been already defined. It supports the following operations:

1. Refresh Summary This operation is used to get an updated status information about the Question.
4.12. QI Editor Components

Figure 4.12: Question Summary component of QI Editor

2. **View Indicator**  The user can select an already finalized indicator which is in memory and select this operation to view its properties in a pop up window.

3. **Edit Indicator**  The user can select an already finalized indicator and use this option to load it back in the editor for editing. In that case, the indicator is deleted from memory and loaded into the editor for re-editing. It must be saved after making changes else all changes are lost.

4. **Delete Indicator**  Sometimes the user can come across a situation when he decides to delete a previously defined indicator and can achieve his goal by this operation. The user needs to select an indicator and use this operation and the indicator gets deleted from memory. Then the “Refresh Summary” can be used to get the updated status.

4.12.3 Indicator Information

Figure 4.13 shows the Indicator Information component. This component exclusively deals with the property definition of an Indicator. Every Indicator must have some basic properties so that relevant data can be fetched from the database. These properties are namely Source, Action, Platform and Minor. It supports the following operations:

1. **Visualize**  Selecting this operation will generate a graph using the properties specified above.

2. **Finalize**  This will save the Indicator with all its properties to memory and then a new Indicator definition process can be started.

4.12.4 Indicator Properties & Summary

Figure 4.14 shows the Indicator Properties & Summary component. This component is pivotal as far as the definition of an Indicator is concerned. Here the user can add various kinds of filters to filter and refine the basic data fetched from database using the basic properties specified in “Indicator Information” component. Filters can be Attribute Filters (to search data

8 Refer to the Data source in Figure 3.2 for more information
using property values), User Filters (to search data for a particular user), Time Filters (to search data for a particular time range) or Session Filters (to search data matching some specific session values). Appropriately, the summary of different kinds of filters applied can be viewed by the user in the “Filter Summary” section. The graph options for the indicator like Graph type (Bar chart, Pie chart etc.) and the library to be used for graph generation can be specified in the “Graph Options” section. The graph or visualization for the current indicator can be viewed in the “Graph Preview” section. The “Indicator Summary” section offers a comprehensive summary for the current indicator.

Using the functionalities provided by the “QI Editor” the user can effectively create new Questions in the run time of the system. The pool of Questions and Indicators will increase with due usage of the system. This functionality is what differentiates this thesis work from all existing
systems which are bounded by the limitation of pre-defined indicators which cannot be cus-
tomized at run-time.

4.13 Summary

In this chapter, the technologies used for the development of the system, the architecture used and the components as well as functionalities provided by the Question-Indicator editor have been examined. The next chapter deals with user evaluation and discusses the results obtained from user survey in detail.
5 Evaluation

“I like to regard myself as someone who’s capable of critical thought, that is to say, who can evaluate claims.”

–Bill Nye

In this chapter, the evaluation of the system developed has been described. Evaluation is done to ensure that the system is both usable and useful. In this thesis work, the aim was to develop the architecture and focus more on the functionalities. Therefore only a minimal UI was developed to ensure that the user could use the functionalities provided by the system. Based on the functionalities provided a set of evaluation questionnaire was developed and used to evaluate the work.

5.1 Evaluation Questionnaire

The evaluation has been broadly divided into four sections namely:

1. User background and familiarity with LA systems.
2. User expectation from a LA system.
3. System usefulness evaluation.
4. System usability evaluation.

During the process of user evaluation, the user was briefly introduced to the concepts of the system. They were also given a preliminary information about what they could achieve with the system along with good concrete examples. The system was evaluated with two teachers, three research assistants and seven students.

The evaluation questionnaire has been included in Appendix 4. The evaluation method and results has been described in detail in the following sections.
5.1.1 User Background Evaluation

In the background and expectation evaluation, about 75% of the users had idea about LA. And all of them supported the idea to have a LA system in RWTH which can be accessed by both students and teachers online. This shows that the users are desirous of trying LA in their respective courses to gain useful insights on how they perform in comparison to other students and similar kind of useful as well as interesting stuffs. But only about 33% of the users had actually used a LA system with real time data and framed their own questions in a LA system. Figure 5.1 shows the user background survey results.

![Figure 5.1: User Background Survey Result](image)

5.1.2 User Expectations Evaluation

The user expectation section specifically emphasized on the features which the users generally would like to use in a LA system. About 90% of the users wanted to formulate their own questions and indicators as well as preview them. Over 90% of the users wanted to select different combination of data, select visualization types, modify existing indicators to create new ones and delete their indicators. About 88% of the users wanted to customize existing indicators available in the system. However only 68% of the users wanted to restrict themselves to using available indicators in the system. Figure 5.2 shows the user expectation survey results.

![Figure 5.2: User Expectation Survey Result](image)

5.1.3 System Usefulness Evaluation

The LA usefulness quality parameters have been taken from work done by Chatti, et al [15]. The parameters are:
5.1. Evaluation Questionnaire

The questions are formulated in a way that is appropriate to capture data and help deduce information about these parameters. The questions in the “The System Usefulness Measurement” section of the user questionnaire in Appendix 4 specifically deal with the functionalities provided by the system. So in essence these questions capture the “usefulness” aspect. Specific questions dealing with definition of indicators, combining them, adding filters, capture “personalization”. The questions which deal with if the user was able to execute the indicator, get a graphical visualization, capture the “monitoring”, “reflection” and “awareness” aspects.

There were ten questions in total and a maximum score of 5 for each question. So from the maximum score of 50, the average of user rating was 42.75 which is approximately 85%.

Figure 5.3 shows the average percentage of user rating on system usefulness. Almost all the question ratings were over 80%. Questions like “Define visualization type for indicator”, “Preview Indicator” and “Reuse existing Indicators” had the least score of 80%. Using functionalities like “Define visualization type for indicator” or “Preview Indicator” was not complicated as it involved a single click. These options were clearly visible in the UI. However reusing existing indicators involved a bit complicated scenario of searching for an existing indicator, view its properties and then load it into editor for reusing. The user had to perform multiple operations to achieve this functionality and hence this particular question received the least score from the users.

Figure 5.4 shows the average percentage of user rating on system usefulness classified into different usefulness quality parameters. Personalization is at 84.5% while monitoring, reflection and awareness are over 87.5% respectively.
5.1.4 Usability Evaluation

In this section, a background information on usability of a product has been discussed and then the discussion proceeds to the system's usability evaluation results.

5.1.4.1 Usability of a System

Usability is a function of the ease of use (including learnability when relevant) and the acceptability of the product and will determine the actual usage by a particular user for a particular task in a particular context. Ease of use determines whether a product can be used, and acceptability whether it will be used, and how it will be used. Ease of use in a particular context is determined by the product attributes, and is measured by user performance and satisfaction. The context consists of the user, task and physical and social environment [7]. The relationship between these factors is shown in Figure 5.5.

The product attributes which contribute to usability include the style and properties of the interface, the dialogue structure, the nature of the functionality, and any other relevant properties such as system efficiency and reliability. Measures of attitude and performance provide the criteria which determine whether the design of the attributes is successful in achieving usability [7].

Usability is not a quality that exists in any real or absolute sense. Perhaps it can be best summed up as being a general quality of the appropriateness to a purpose of any particular
5.1. Evaluation Questionnaire

Since usability is itself a moveable feast, it follows that measures of usability must themselves be dependent on the way in which usability is defined [10]. It is possible to talk of some general classes of usability measure; ISO 9241-11 suggests that measures of usability should cover [10]:

1. effectiveness (the ability of users to complete tasks using the system, and the quality of the output of those tasks).
2. efficiency (the level of resource consumed in performing tasks).
3. satisfaction (users’ subjective reactions to using the system).

5.1.4.2 The System Usability Scale (SUS)

The demands of evaluating usability of systems within an industrial context mean that often it is neither cost-effective nor practical to perform a full-blown context analysis and selection of suitable metrics. Often, all that is needed is a general indication of the overall level of usability of a system compared to its competitors or its predecessors. Equally, when selecting metrics, it is often desirable to have measures which do not require vast effort and expense to collect and analyse data. In response to these requirements, a simple usability scale was developed. The System Usability Scale is a simple, ten-item scale giving a global view of subjective assessments of usability[10]. It consists of a 10 item questionnaire with five response options for respondents; from Strongly agree to Strongly disagree. It allows the owners to evaluate a wide variety of products and services, including hardware, software, mobile devices, websites and applications. The benefits of using SUS are:

1. It is a very easy scale to administer to participants.
2. It can be used on small sample sizes with reliable results.
3. It is valid - it can effectively differentiate between usable and unusable systems.

5.1.4.3 Score Interpretation with SUS

Interpreting scoring can be complex. The participant’s scores for each question are converted to a new number, added together and then multiplied by 2.5 to convert the original scores of 0-40 to 0-100. Though the scores are 0-100, these are not percentages and should be considered only in terms of their percentile ranking. SUS yields a single number representing a composite measure of the overall usability of the system being studied [10]. The detailed score interpretation procedure is outlined below:

1. Scores for individual items are not meaningful on their own.
2. Firstly the score contributions from each item are summed up. Each item’s score contribution will range from 0 to 4. For items 1, 3, 5, 7, and 9 the score contribution is the scale position minus 1. For items 2, 4, 6, 8, and 10, the contribution is 5 minus the scale position.
3. Then the sum of the scores is multiplied by 2.5 to obtain the overall value of SUS.

5.1.4.4 Usability Evaluation Results

For usability evaluation, the SUS survey method has been adopted. The questions are designed to capture the effectiveness of the UI, users comfort level in using the UI. It also shows
the steepness of the learning curve which a user must go through to successfully use the UI. The results of this evaluation is shown in Figure 1 in Appendix 4. The usability scale of the system is approximately SUS score of 53. Users wanted the option of defining custom indicators but did not want to go through a complicated process of indicator definition. They desired the process to be simplified in just a few clicks. Moreover, majority of the users wanted a more intuitive way of on-screen guidance during the indicator definition process.

But since the focus of this work is mainly on the architecture and functionalities hence with this result the system can be used as a early pre-view. This also paves the road to future work, where work will be done to improve the UI so that the system can be easily used by masses.

5.2 Evaluation Results

Evaluation results demonstrate that majority of the users liked and appreciated the concept on which the system has been adopted. They liked the features offered by the system like formulation of questions and indicators, flexibility of visualization type selection, previewing capabilities together with aggregation of simple indicators to build composite ones. However, majority of the users suggested that the UI should be more intuitive. The indicator definition process should be made more simple and more innovative user notification mechanism should be implemented which would guide the user through the indicator definition process. Figure 5.6 shows the overall results of user evaluation with regards to usefulness and usability. It is obvious from the evaluation results that future work involving improvements in UI is necessary after which the system can be deployed. In the future, some important evaluations like how the tool is helping to improve the teaching and learning process can be determined to assess the practical usefulness of the tool.
5.3 Summary

In this chapter, the user evaluation of the system has been discussed. System usefulness has been greatly appreciated. Hence it can be concluded that personalization in LA system has been achieved which was the sole objective of this thesis work. The next chapter formally states the conclusions derived from this thesis work.
6 Conclusion and Future Work

“To succeed, jump as quickly at opportunities as you do at conclusions.”

—Benjamin Franklin

This chapter summarizes the whole process of this thesis work. It also discusses about possible future work which can be leveraged on the basis of this work.

6.1 Conclusion

The developed system allows the stakeholders to get involved in the indicator definition process. This makes the system user driven and also contributes significantly to the personalization of LA. The development was done based on the conceptual approach initially conceived. The system is basically a spring based web application and drools has been used as a rule engine for indicator generation. The user can interact with the UI of the system to define a question and some indicators. For every indicator, the type of data that the user is interested to visualize can be specified. For refinement of the data fetched, filters can be added. All these selections are passed to the Indicator Engine where they are parsed and an equivalent indicator is generated based on the rules present in the rule engine. The user gets an opportunity to select the visualization type for every indicator. The indicator can be previewed and changes can be made repeatedly with respect to data fetched and filter specifications. Multiple indicators can be combined to create new composite ones. Instead of defining every indicator from scratch, the user can customize an existing one into a new one. After all selections and association of indicators with a question it can be saved for later retrieval and execution. After development the system was subjected to user evaluation.

For evaluation, the two most important criteria of usefulness and usability were considered. A set of questions was prepared evaluating the user expectations, usefulness and usability of the system. From the user study, it was clear that majority of the users wanted personalization in LA. They wanted to define their own questions, indicators, combine multiple ones and modify existing ones. All most all the users liked the concept of the system and the functionalities it offered. From the user evaluation results it is easy to answer the research questions which was the aim of this thesis work:
How can we make a LA system more dynamic and add elements of Personalization to it?

To add elements of personalization to LA, the system is designed to allow the user to formulate their questions and indicators. The users were asked to perform a series of tasks with the system extensively covering all the functionalities provided by the system. The users liked the personalization aspects introduced in the system and the overall usefulness of the system was about 90%.

From the evaluation results, it can be concluded that most of the users found it easy to define an indicator, define visualizations for the indicator, combine multiple indicators into composite ones. All these task questions were voted over about 80%. Only one of the task which involved reusing an existing indicator form the system database and modifying it into a new one was slightly complicated as evident from the user votes. This specific question has an average of 80% rating. From the concepts introduced in this thesis work, its actual implementation and user survey it can be concluded that a LA system can be more dynamic and personalized.

Is such a system useful as well as usable?

The user evaluation results clearly suggests that most users want a personalized LA system. This is evident from the user expectation survey results where almost 90% of the users were in favour of such a system. The developed system is useful as most of the users voted in favour of its usefulness.

For usability, the SUS scale was used and the score stood at 54%. The system cannot be used by people especially from non-technical background. Users want a nice, clean UI which would simplify the indicator definition process. The current UI is not very intuitive as the thesis solely aimed in development of the system concepts and architecture. The system can be made usable in subsequent works.

Thus the research questions are answered from the user study. It is possible to introduce personalization in LA and such a system is useful at the moment.

6.2 Future Work

Indicator Engine is a rule based tool-kit designed to permit users to define new indicators during the runtime of a LA system. It also allows users to customize exiting indicators. It is one of the core components of the open learning analytics platform. Currently it has a limited set of indicator operations and visualization techniques available at its disposal. Work has been initiated to enhance its functionality and integrate it with the open learning analytics platform. They have been described in detail in the sections below.
6.2. Future Work

6.2.1 Enhancement of User Interface

After analysing the results of user study as well as the suggestions made by the users for improvement, it becomes very clear that the UI of the system needs to be improved. A thesis work is already in progress which would develop a cleaner and much simpler UI operating on the code developed in this thesis work.

6.2.2 Enhancement of Indicator Operations

The system can be further extended to generate indicators capable of performing more complex operations like finding correlation between two different data sets or performing regression analysis. A detailed documentation has been prepared to aid the future development process.

6.2.3 Modularity Framework for Open Learning Analytics Platform

Indicator Engine is one of the components of the open learning analytics platform. It would be integrated into the platform. A further work to develop an architecture for integrating all the components of the system is in progress. The modularity of the open learning analytics platform is one of its core design focuses. For this end, it is necessary to define a modularity framework for allowing researches and developers to extend the core functionality in a consistent, flexible, simple, standardized and secure manner. The framework aims to provide:

1. A well constructed API that is easy to implement, easily importable to projects and with meaningful error handling.
2. A RESTful API with focus on security and usability.
3. A way for developers and researchers to create new analytics modules via API.
4. A way for developers and researchers to be able to integrate new analytics methods via API.

At the core of the system lies an analytics engine. It is in charge of performing the analysis and executing indicator queries and getting data according to the selected methods as well as creating the visualization of the data with the selected mechanism. The indicators are created or selected from a specialized component controlled by the user and provide indicator/goal/visualization triads. These represent questions that stakeholders make the system for it to analyse data and present it in a relevant manner. As part of its functioning, the Analytics Engine provides means to connect multiple analytics methods and analytical modules. This interconnection is the focus of the future work.

The framework core is a modularity engine inside the OLA platform. The engine will be able to process API calls from both developers and the analytics Engine itself. The modularity engine provides an external API for developers to create new analytical modules and analytical methods as well as revising them and updating them if required. Additionally, the analytics engine will have an internal API that will allow it to interact with the repository of analytical methods and analytical modules. The core idea is to decouple the dependency of the analytic...
engine from the extensible parts of the OLA platform. A component diagram of the framework is shown in Figure 6.1.

![Component Diagram of Framework](image)

**Figure 6.1: Modularity Framework: Component Diagram [64]**

6.2.3.1 Analytical Modules

Analytical modules need to be register via XML and an API will be provided for JAVA interpretation of the object as a POJO/Bean internally. Means for the analytics engine to interact with the modules will be implemented through the API through interfaces to support the flexibility of the architecture.

6.2.3.2 Analytical Methods

The addition of new analytical methods will be handled by the developers via the implementation of an interface in Java. This will also allow the internal API to create an abstraction that will be made available to the analytics engine.

6.2.4 Modular Visualization Framework for the Open Learning Analytics Platform

The visualization capabilities of the current system can be improved to include wide range of charting engines and charting types. This would allow the user to visualize the data in
many different ways. This would be handled by a separate component called “visualizer”. The visualizer component in the open learning analytics platform is responsible for providing easy mechanisms to manage, add, and remove visualization techniques such as Google Charts, D3/D4, jQGraph, Dygraphs, jqPlot along with the type of visualization (e.g. bar chart, pie chart, line chart, etc.) supported by each technique. An adapter is required for each visualization technique to transform the data format used in the analytics engine to the indicator visualization code to be rendered on the client application (e.g. dashboard, HTML page, widget) [16].

The indicator generated by the Indicator Engine will communicate with the visualizer which will then decide how the data fetched would be visualized. It would decide the visualization framework and the graph options to use to render the data. The flow of communication between Indicator Editor (while defining an indicator) and the visualizer is shown in Figure 6.2. It would provide suggestions of possible graph options for the indicator being defined. Figure 6.3 represents how data fetched with an indicator execution is visually presented to the user.

![Figure 6.2: Future Work: Flow of formulating question and its visualization technique [3]](image)

### 6.3 Summary

In this thesis work, it has been primarily strived to develop “Indicator Engine” which is one of the major components of the open learning analytics platform. Open learning analytics strives to develop a common platform where different stakeholders can participate to analyse their dataset. These stakeholders will naturally have different objectives. Hence the concept of developing a LA system with a pre-defined set of indicators will not suffice. Therefore a way has been devised to help the users define their own indicators dynamically. This introduces elements of personalization to the LA system. The concept and usefulness of the system
has been greatly appreciated. With the completion of all the discussed future work, the open learning analytics platform together with its indicator engine, analytics engine and visualization framework will be able to satisfy the diverse needs of a diverse range of stakeholders using the system. This would definitely satisfy the constraints of open learning analytics.
List of Figures

2.1 LA reference model .................................................. 6
2.2 Learning Analysis Process ........................................... 7
2.3 Open Learning Analytics Platform .................................. 10
2.4 Architecture of Academic Analytics Tool .......................... 13
2.5 Architecture of LeMo .................................................. 13
2.6 Indicator execution process ......................................... 14
2.7 Analytics Workbench in Action ..................................... 15

3.1 Abstract Architecture .................................................. 21
3.2 Data Source ER Model ................................................. 22
3.3 Conceptual Diagram of Deployed Application .................... 23
3.4 Overall Use Case Scenario ............................................ 25
3.5 Indicator definition: Use Case Scenario ............................ 25
3.6 Filter addition to an Indicator: Use Case Scenario ............... 26
3.7 Modify existing indicator: Use Case Scenario ..................... 27

4.1 Indicator Engine Architecture ........................................ 31
4.2 System User ER Model ............................................... 35
4.3 Question & Indicator ER Model ...................................... 38
4.4 Sequence Diagram: User and System Interaction ................ 39
4.5 Different Objects of the class: “Student” ......................... 40
4.6 Question-Indicator Editor Interface ................................. 43
4.7 View Existing Questions in the Database ........................ 49
4.8 Question Detail Listing ............................................... 50
4.9 Question Execution .................................................... 50
4.10 Question-Indicator Editor ............................................ 51
4.11 Question Information component of QI Editor ................... 52
4.12 Question Summary component of QI Editor ..................... 53
4.13 Indicator Information component of QI Editor .................. 54
4.14 Indicator Properties & Summary component of QI Editor .... 54

5.1 User Background Survey Result .................................... 58
5.2 User Expectation Survey Result ..................................... 59
5.3 Average User Ratings on System Usefulness ..................... 60
5.4 User Ratings based on Usefulness Metrics ........................ 61
5.5 Determinants of usability ............................................. 62
Chapter 6. List of Figures

5.6 Overall User Ratings .................................................. 63
6.1 Modularity Framework: Component Diagram .......................... 68
6.2 Flow of formulating question and its visualization technique .............. 69
6.3 Flow of getting visualization for an indicator data request ............... 70
1 User Ratings on Usefulness .............................................. 91
Bibliography


Chapter 6. Bibliography


Appendices
Source Codes

This chapter provides various source code snippets which were used in development. These snippets are only for demonstration purposes only and they are not complete by themselves.

1 Question: Object Relational Mapping

We have designed a m:n relationship between Question and Indicators in the system. This relationship mapping was implemented using Hibernate. Following is the relevant code for the Question Object.

```java
@Entity
@Table(name = "gla_Question")
public class GLAQuestion implements Serializable {
    @Id
    @Column(name = "question_id", unique = true, nullable = false)
    @GeneratedValue(strategy = IDENTITY)
    @Expose
    private long id;

    @Column(name = "question_name", nullable = false, unique = true)
    @Expose
    private String question_name;

    @Column(name = "indicators_num", nullable = false)
    @Expose
    private int indicators_num;

    @ManyToMany(cascade = { CascadeType.ALL })
    @JoinTable(name = "gla_Question_Indicator", joinColumns = { @JoinColumn(name = "question_id") }, inverseJoinColumns = { @JoinColumn(name = "indicator_id") })
    private Set<GLAIndicator> glaIndicators = new HashSet<GLAIndicator>();

    @OneToOne(fetch = FetchType.LAZY, cascade = CascadeType.ALL,
            mappedBy = "glaQuestion", orphanRemoval = true)
    private GLAQuestionProps glaQuestionProps;
}
```
2 Indicator: Object Relational Mapping

Following is the relevant code for Indicator Object.

```java
@Entity
@Table(name = "gla_Indicator")
public class GLAIndicator implements Serializable {

@Id
@Column(name = "indicator_id", unique = true, nullable = false)
@GeneratedValue(strategy = IDENTITY)
@Expose
private long id;

@Column(name = "indicator_name", nullable = false, unique = true)
@Expose
private String indicator_name;

@Column(name = "short_name", nullable = false)
@Expose
private String short_name;

@Column(name = "hql", nullable = false, columnDefinition="TEXT")
@Expose
private String hql;

@OneToOne(fetch = FetchType.LAZY, cascade = CascadeType.ALL,
  mappedBy = "glaIndicator", orphanRemoval = true)
@Expose
private GLAIndicatorProps glaIndicatorProps;

@ManyToMany(mappedBy = "glaIndicators")
private Set<GLAQuestion> glaQuestions = new HashSet<GLAQuestion>();
}
```

3 Time Filter: Implementation

For Every Indicator, the user can add filters to filter out irrelevant data. One such example is Time Filter which enables the user to frame an indicator which can search for data within a particular time range. Following is the code for the implementation of Time Filter.

```java
public String processTime(List<TimeSearchSpecifications> timeSearchSpecifications, String filter){
  String hibernateQuery=" ";
  String hibernateExactTime =" AND glaEvent.timestamp IN (";
  String hibernateRangeTime =" ";
  String type = null;
  List<String> time = new ArrayList<>();
  int counterTime = 0;
  int counterRangeTime = 0;
  for(TimeSearchSpecifications timeSpec : timeSearchSpecifications){
    time = timeSpec.getTimestamp();
  }
  return hibernateQuery + time + hibernateExactTime + filter + hibernateRangeTime + type;
}
```
type = timeSpec.getType();
if(type.equals("EXACT")) {
    if(counterTime == 0) {
        Date date = new Date(Long.parseLong(time.get(0))); //
        *1000 is to convert seconds to milliseconds
        SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd
        HH:mm:ss"); // the format of your date
        String formattedDate = sdf.format(date);
        hibernateExactTime += "'"+formattedDate+"'";
        counterTime++;
    }
    if(counterTime >= 1) {
        Date date = new Date(Long.parseLong(time.get(0)));
        SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd
        HH:mm:ss");
        String formattedDate = sdf.format(date);
        hibernateExactTime +=","+formattedDate+",";
        counterTime++;
    }
}
if(type.equals("RANGE")) {
    if(counterRangeTime == 0) {
        Date startDate = new Date(Long.parseLong(time.get(0)));
        SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd
        HH:mm:ss");
        String startDateFormattedDate = sdf.format(startDate);
        Date endDate = new Date(Long.parseLong(time.get(1)));
        String endDateFormattedDate = sdf.format(endDate);
        hibernateRangeTime += "AND glaEvent.timestamp BETWEEN'
        +startFormattedDate+" AND "+
        +endFormattedDate+"'";
    }
    if(counterRangeTime >= 1) {
        Date startDate = new Date(Long.parseLong(time.get(0)));
        SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd
        HH:mm:ss");
        String startDateFormattedDate = sdf.format(startDate);
        Date endDate = new Date(Long.parseLong(time.get(1)));
        String endDateFormattedDate = sdf.format(endDate);
        hibernateRangeTime += " AND glaEvent.timestamp BETWEEN'
        +startFormattedDate+" AND "+
        +endFormattedDate+"'";
    }
}
hibernateExactTime += " ");
if(counterTime > 0)
    hibernateQuery += hibernateExactTime;
if(counterRangeTime > 0)
    hibernateQuery += hibernateRangeTime;
return hibernateQuery;
"

4 Query Geneation Rule Example

The user selected options and parameters are parsed by Drools to generate the final Hibernate Query for an indicator. The following code snippet demonstrates one such scenario:

dialect "mvel"

rule "When Only Entity Filtering (Default Rule) is Present"
when
    EntitySpecification (userSpecifications.isEmpty() == true, 
    timeSpecifications.isEmpty() == true, 
    sessionSpecifications.isEmpty() == true, 
    entityValues.size() == 1, 
    entityValues.get(0).geteValues() == "ALL")
$pFilter : ProcessUserFilters()
$userParameter : EntitySpecification($source : selectedSource, 
    $platform : selectedPlatform, 
    $action : selectedAction, 
    $major : selectedMajor, 
    $minor : selectedMinor, 
    $type : selectedType, 
    $key : entityValues, 
    $pobject : persistenceObject, 
    $filter : filteringType, 
    $robject : retrievableObjects)
then
$userParameter.setHql("SELECT " + $robject + " FROM " + $pobject + " " 
    WHERE glaEvent.action ='"+action+"'")
    +" AND glaEvent.source IN "+$pFilter.
    processSource($source,$filter) + " 
    AND glaEvent.platform = '"
    +$platform+"'"+" AND glaEvent.
    glaCategory.major = '"$major+"'
    +" AND glaEvent.glaCategory.minor = '"+ 
    $minor+"'"+" AND glaEvent.
    glaCategory.type = '"
    +$type+"'"+" AND key = '"+ $key.get(0).
    getKey()++"";
end
User Evaluation Questionnaire
A Rule-Based Indicator Definition Tool for Personalized Learning Analytics

User Evaluation

* Required

1. **Age** *
   *Mark only one oval.*
   - [ ] < 25
   - [ ] 25 - 40
   - [ ] > 40

2. **Gender** *
   *Mark only one oval.*
   - [ ] Male
   - [ ] Female

3. **Field of Study** *

4. **Occupation** *

5. **Nationality** *

---

**User Background Information**

Are you familiar with Learning Analytics and similar technologies?

6. **Are you aware of Learning Analytics (LA) ?** *
   *Mark only one oval.*
   - [ ] Yes
   - [ ] No

7. **Do you think it is useful to have a LA System in RWTH ?** *
   *Mark only one oval.*
   - [ ] Yes
   - [ ] No
8. Have you used Learning Analytics on real time data? *
   *Mark only one oval.
   ○ Yes
   ○ No

9. Have you ever formulated your own Indicator in a LA System? *
   *Mark only one oval.
   ○ Yes
   ○ No

User Expectation
What do you expect from a LA system?

10. I want to define my own Questions & Indicators. *
    *Mark only one oval.

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
</table>
| Strongly disagree | | | | | Strongly agree

11. I want to use the pre-existing Questions & Indicators only. *
    *Mark only one oval.

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
</table>
| Strongly disagree | | | | | Strongly agree

12. I want to reuse existing Questions & Indicators to define my own ones. *
    *Mark only one oval.

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
</table>
| Strongly disagree | | | | | Strongly agree

13. I would like to have the option of selecting which data to be used for my Indicator. *
    *Mark only one oval.

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
</table>
| Strongly disagree | | | | | Strongly agree

14. I would like to have the option of specifying multiple filters to my Indicator.
    *Mark only one oval.

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
</table>
| Strongly disagree | | | | | Strongly agree
15. **I would like to select the visualization type (bar chart, pie chart etc.) of my Indicator.**

*Mark only one oval.*

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Strongly disagree</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

16. **I would like to combine two or more existing Indicators to a new Indicator.**

*Mark only one oval.*

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Strongly disagree</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

17. **I would like to have the option to edit/delete my Indicators.**

*Mark only one oval.*

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Strongly disagree</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

18. **I would like to have the option to preview my Question/Indicators before saving them.**

*Mark only one oval.*

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Strongly disagree</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Task Definition: The System Usefulness Measurement**

How useful is the system?

19. **I was able to define my own Question and Indicators easily.**

*Mark only one oval.*

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Strongly disagree</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

20. **I was able to define filters for my Indicator easily.**

*Mark only one oval.*

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Strongly disagree</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
21. I was able to define visualization type for my Indicator easily. *
   *Mark only one oval.

   1 2 3 4 5

   Strongly disagree   Strongly agree

22. I was able to preview my Indicator easily. *
   *Mark only one oval.

   1 2 3 4 5

   Strongly disagree   Strongly agree

23. I was able to search for a specific indicator easily. *
   *Mark only one oval.

   1 2 3 4 5

   Strongly disagree   Strongly agree

24. I was able to reuse existing Indicators easily. *
   *Mark only one oval.

   1 2 3 4 5

   Strongly disagree   Strongly agree

25. I was able to edit/delete my Indicators easily. *
   *Mark only one oval.

   1 2 3 4 5

   Strongly disagree   Strongly agree

26. I was able to combine Indicators easily. *
   *Mark only one oval.

   1 2 3 4 5

   Strongly disagree   Strongly agree

27. I was able to visualize existing Questions/Indicators easily. *
   *Mark only one oval.

   1 2 3 4 5

   Strongly disagree   Strongly agree
28. I liked the way the "User Notifications" are presented on the screen. *
   Mark only one oval.

   1 2 3 4 5

   Strongly disagree ☐ ☐ ☐ ☐ ☐ Strongly agree

The System Usability Scale: Questions
For Understanding Usability of the System.
All items should be checked. If a respondent feels that they cannot respond to a particular item, they should mark the centre point of the scale.

29. I think that I would like to use this system frequently. *
   Mark only one oval.

   1 2 3 4 5

   Strongly disagree ☐ ☐ ☐ ☐ ☐ Strongly agree

30. I found the system unnecessarily complex. *
   Mark only one oval.

   1 2 3 4 5

   Strongly disagree ☐ ☐ ☐ ☐ ☐ Strongly agree

31. I thought the system was easy to use. *
   Mark only one oval.

   1 2 3 4 5

   Strongly disagree ☐ ☐ ☐ ☐ ☐ Strongly agree

32. I think that I would need the support of a technical person to be able to use this system. *
   Mark only one oval.

   1 2 3 4 5

   Strongly disagree ☐ ☐ ☐ ☐ ☐ Strongly agree

33. I found the various functions in this system were well integrated. *
   Mark only one oval.

   1 2 3 4 5

   Strongly disagree ☐ ☐ ☐ ☐ ☐ Strongly agree
34. I thought there was too much inconsistency in this system. *

Mark only one oval.

1 2 3 4 5

Strongly disagree ☐ ☐ ☐ ☐ ☐ Strongly agree

35. I would imagine that most people would learn to use this system very quickly. *

Mark only one oval.

1 2 3 4 5

Strongly disagree ☐ ☐ ☐ ☐ ☐ Strongly agree

36. I found the system very cumbersome to use. *

Mark only one oval.

1 2 3 4 5

Strongly disagree ☐ ☐ ☐ ☐ ☐ Strongly agree

37. I felt very confident using the system. *

Mark only one oval.

1 2 3 4 5

Strongly disagree ☐ ☐ ☐ ☐ ☐ Strongly agree

38. I needed to learn a lot of things before I could get going with this system. *

Mark only one oval.

1 2 3 4 5

Strongly disagree ☐ ☐ ☐ ☐ ☐ Strongly agree

Overall Feedback
How did you like the system in general?

39. I am satisfied with the functionalities provided by the system. *

Mark only one oval.

1 2 3 4 5

Strongly disagree ☐ ☐ ☐ ☐ ☐ Strongly agree
40. I am satisfied with the ease of use of the functionalities. *
Mark only one oval.

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Strongly disagree</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

41. Suggestions for Improvement. *

________________________________________________________________________
________________________________________________________________________
________________________________________________________________________
________________________________________________________________________
________________________________________________________________________
User Evaluation Results

![Table of User Ratings on Usefulness](image)

**Figure 1**: User Ratings on Usefulness