Enhancements to Google Course Builder: Assessments Visualisation, YouTube Events Collector and Dummy Data Generator

Miriam Marciel, Foivos Michelakis, Rodérick Fanou, and Pedro J. Muñoz-Merino, Member, IEEE

Abstract—This paper presents some extensions to the functionality of Google Course Builder (GCB) for improving its learning analytics support. The contributions of our work are the following: we add the functionality of detecting and storing YouTube events, visualise complex data related to the performance of the students, and provide automatic suggestions to the teachers regarding which specific parts of material need revision. Finally, we test our approach by implementing a dummy data generating program, which creates a big number of fictitious students who can interact with the platform in a fashion identical to real students.

Keywords—Google Course Builder, YouTube events, dummy data generator, learning analytics, visualisation

I. INTRODUCTION

Platforms that provide online courses to a large demographic public are widely used at present. As a matter of fact, Google Course Builder (GCB)\textsuperscript{1} is an open source platform that enables the delivering of educational courses through the Web. GCB allows dividing a course into lessons, uploading a set of videos that can be watched by students or creating a collection of assessments. Courses in GCB can be made available to anyone on the Web.

Furthermore, learning analytics is an emerging discipline for analyzing and understanding better the learning process based on the different data collected. The application of learning analytics techniques is very important in platforms like GCB. In such platforms, there can be many teachers and students thus generating a big amount of data, and it is crucial to understand the learning process and try to improve it. There are studies that describe the expectations of stakeholders regarding learning analytics, such as [1].

The process of learning analytics is usually divided into different phases. First, there is the collection of students’ events, and interactions (low level data). Next, this low level data should be processed in order to obtain intelligent information (since raw data are usually useless). The third phase consists of the visualisation of this data, whereas the last one consists of specific actions such as recommendations.

Each of the commented learning analytics phases presents interesting challenges. The collection of data implies the definition of the entities, their fields, and their relationships in a way that is complete so that the different events and user interactions can be retrieved. It should also be easy to process. In addition, there are other issues to solve such as the format used to store the data, or interoperability issues in order to make the solution work in a distributed oriented environment, and be general enough to adapt to different educational settings and platforms.

The processing of low level data to obtain intelligent information is a crucial part of the learning analytics process. Depending on the specific educational platform, different information can be retrieved. For example, a platform such as Khan Academy can have parametric exercises that a student may answer correctly several times. If he/she does so without viewing hints during his/her first attempt, he/she obtains proficiency in a skill. On the other hand, GCB can have an assessment composed of several exercises with the overall grade depending on the number of correct answers. Then, the different features and semantics of the platform for the assessment functionality will influence the type of data that can be collected, as well as the higher level information that can be inferred from it. In addition, among all the information that can be obtained, it is important to select the one that is more useful for teachers, students and other stakeholders, and communicate it in a precise and clear way.

Visualisation is also a challenge, since there are many ways to visualise the information. Usually, usability studies are required among students and teachers. Finally, the interpretation of the information to give proper advice to teachers and students is a very difficult issue, and typical actions are recommendations based on many aspects like pedagogical theories, etc. Other possible actions that are enabled by learning analytics are explained in [2].

In this work, we address some aspects of learning analytics for the GCB platform. The initial functionality of GCB has been extended to include these enhancements:

- Visualisation of assessment data. Low level data about students’ interactions with assessment tests are already stored in the Datastore of GCB: data about the specific
students’ answers, and their different attempts in the different assessments. In this work, some visualisations are proposed so that teachers can view useful information. In addition, some recommendations are proposed to try to advice teachers e.g. on badly designed exercises.

- Collector of YouTube events. Since GCB supports YouTube videos that can be watched by students, the retrieval of low level data regarding videos is an important issue. Data such as the events related to the press of different video buttons (e.g. forward or rewind) are interesting, since knowing the way that students watch videos can be useful for the learning process. In this work, a solution is proposed to store this data so that other tools can infer and represent this low level data.

An orthogonal issue to learning analytics is the testing and the evaluation of the developed solutions. In many occasions, when developing a solution, there are not real data of students interacting with a course, making the testing and evaluation difficult. In order to address this issue, this work presents a dummy generator of students for GCB that can create fictitious students and populate the database simulating real students. In this way, fictitious YouTube events are generated, as well as fictitious assessment events. The developed tools for learning analytics can be tested visualising assessments of fictitious students or collecting data from fictitious YouTube events. As this is an orthogonal issue to learning analytics, this module might be used in the future for other purposes apart from just learning analytics.

This work also shows an architecture for integrating all of the commented components: dummy generator of students, visualisation and recommendations related to assessments, and collector of YouTube events. The different elements are shown as well as their relationship, and the place they occupy in the GCB infrastructure. We release the final version of our code on the following link².

The remainder of this paper is structured as follows. Section II discusses related work. In section III, we describe the proposed architecture of GCB. Sections IV, V and VI present a technical overview of the assessments extensions, the YouTube events collector, and the dummy data generator, detailing their implementation, features, and performance. Finally, sections VII and VIII draw conclusions and give elements for future directions.

II. RELATED WORK

The collection of low level data is an interesting issue with many challenges. This task is essential so that other phases of the learning analytics process can take place. In this direction the Contextualized Attention Metadata (CAM) solution [3] defines a model for retrieving data related to different students’ events. The model includes information such as the time when the interaction is created, the resources involved, the application used, or the context in which the interaction takes place. The solution is done in a way that is generic enough to retrieve data from many different sources and applications and has been illustrated in many typical cases. An Extensible Markup Language (XML) binding is also proposed [3] for enabling interoperability among different systems. This solution has also been analyzed so that it can work in the Semantic Web [4], formatted in RDF.

By contrast, in the present work, the YouTube events (similar to CAM data) are stored in the GCB database. Indeed, the data stored in the GCB Datastore might be converted into CAM data in the XML format if it would need to interoperate with other systems. The contribution of this work with respect to this issue is to adapt specific YouTube events to a type of data that can further be processed.

There are many assessment systems that generate exercises to students (e.g. [5], [6], [7], [8]). Indeed, there is a specification, IMS-QTI [9], which sets many different possibilities of assessments. The specific functionality of each assessment system determines the type of data that can be collected and information that can be inferred and visualised. In this line, assessment systems can generate hints or not, or implement competition or not. Some of them can have parametric exercises. Some can even have different exercises on the same page, whereas others, only one exercise per page. The work in [10] presented different inferences of higher level information from low level data for the Khan Academy platform, including assessment related information. In our work, one contribution is the analysis of the GCB assessment environment to know the useful information that can be retrieved according to the semantics and functionality of GCB.

There are also different works on visualisation of educational information. Some examples are the following:

- Students’ activities, goal setting or event oriented information [11]
- Social networks [12]
- Resources used, average time spent per resource, or the evolution of the students [13]
- The WebCT platform [14]
- The Moodle platform [15]

Our work makes also a contribution with specific visualisations and recommendations for GCB related to assessments.

III. PROPOSED ARCHITECTURE

GCB is built over the Google App Engine (GAE). The main technologies used for GCB are the python programming language, Javascript, Hypertext Markup Language (HTML), and the Datastore. The work described in this paper is an extension of GCB: on the one hand with new components that are external to GCB; on the other hand using existing components of GCB. Fig. 1 presents our defined architecture including the external elements of GCB (YouTube API, YouTube events collector, dummy students generator, and visualisation module), as well as the Datastore entities that are used or created.

First, we add a YouTubeEvent() entity to the server that collects events (pause, rewind, forward, etc.) from the YouTube videos in the course and these data are stored in the Datastore. To collect these data, we use an Application Programming Interface (API) of YouTube at the students’ browser which

²https://github.com/roderickfanou/Learning_analytics_on_GCB
utilizes Javascript to control the player and notify events. AJAX is used for the communication between the client and the server to store the collected data in the YoutubeEvent() entity.

Next, the Dashboard is enhanced with a visualisation module, which shows information that can be useful for the learning process related to assessments and students, namely from Student(), StudentAnswersEntity() and StudentPropertyEntity() entities. These statistics are used to create recommendations which are also sent to the browser of the teachers (or whoever has administrator access to the course). Graphs and tables are generated for presenting the data in meaningful ways.

Finally, in order to evaluate the proposed architecture, we write a program that generates dummy data. In the following sections, we analyze in detail the different components of this architecture.

IV. ASSESSMENTS EXTENSIONS FOR LEARNING ANALYTICS ON GCB

A course in GCB is divided into units, each unit into lessons and each lesson is a set of videos. At the end of each lesson, the students are allowed to check their knowledge by doing an activity. That exercise, which is not marked, does not influence their evolution in the course contrarily to the assessments. There are three types of assessments which can be tried many times in the default course “Power Searching with Google”:

- The Pre-course assessment: located at the beginning of the course which evaluates the background of the student concerning the topic. Its goal is to give an idea of what the students know coming in, so that what they gain from the course can be better understood.
- The Mid-course assessment: is placed at the middle of the course. It addresses the previous units content.
- The Final-assessment or Post-course assessment: located at the end of the course aims to evaluate all the units.

To pass the course, each student has to complete these three assessments and succeed in getting an average of 50 points out of 100 points in each of them.

It is obvious that the evolution of the students on GCB is tied to their performances during the assessments which depend on their understanding of the videos. It is therefore compulsory for the teachers (or the administrator) to have a deep overview of the students’ assessments statistics.
However, only few statistics are displayed on the administrator Dashboard of GCB version 1.3.1 as far as the learning analytics are concerned: the number of enrolled students, the number of students having completed each assessment, the average score, and the instant at which these statistics are computed and stored in the database (Fig. 2).

In order to help, on the one hand teachers to better know their students, their results, and improve the quality of the course; on the other hand, students to perform better based on the self-reflection, we process more low level data and display more information about assessments.

### A. Data Collection

While exploring the Datastore, we remark that some useful information for the statistics are not actually computed. So, we mainly focus on the answers submitted by each student (stored in StudentAnswersEntity()), the best score obtained by each student for a given assessment (stored in StudentEntity()), and the number of attempts (stored in StudentPropertyEntity()). These data are in JSON format.

### B. Computation

Course builder provides job classes which track job states (started, running, finished) and store job results, none of which is provided by GAE directly. The jobs can be controlled (started, stopped) from the Dashboard interface with the “Re-Calculate now” button (Fig. 2). Among these jobs, the class DurableJob() computes student and assessments statistics and stores its results in the DurableJobEntity() of the Datastore.

In order to improve the performance of the system, we choose to compute our statistics in real time and display some charts of the class overview that the administrator can see in the Dashboard. We add a class AnswersTreatment() which extracts from the Datastore the answers submitted by each student for each assessment (as discussed above). Based on that information, the aggregate number of good answers for each question (AggNumGoodAnsQuest) is inferred, as well as the number of students having completed the corresponding assessment (TotalStudForAss), and the percentage of good answers in the whole class for each question (PerGoodAnsQuest) is processed using (1) and (2).

\[
\text{AggNumGoodAnsQuest} = \sum_{\text{CountGoodAns}=0, i=1}^{\text{TotalStudForAss}} \text{CountGoodAns}
\]

\[
\text{PerGoodAnsQuest} = \frac{\text{AggNumGoodAnsQuest} \times 100}{\text{TotalStudForAss}}
\]

In addition, we write a class StudentTreatment() to retrieve from the Datastore the score (ScoStu[i][Ass]) of each student for the completed assessments and a class PropertyTreatment() for the number of attempts for each assessment (as discussed above). We are therefore able to infer the number of students having succeeded each assessment \(N_{StuSucAss}\) and the mean of attempts for such a category \(\bar{X}_{AtStuSucAss}\) with relations (3) and (4).

\[
N_{StuSucAss} = \sum_{\text{CountStud}=0, i=1, \text{ScoStu[i][Ass]}=50}^{\text{TotalStudForAss}} \text{CountStud}
\]

\[
\bar{X}_{AtStuSucAss} = \frac{\sum_{\text{CountAt}=0, i=1, \text{ScoStu[i][Ass]}=50}^{\text{TotalStudForAss}} \text{CountAt}}{N_{StuSucAss}}
\]

### C. Data Visualisation and recommendations

Moreover, we add a visualisation module which contains, among others, some charts of the class overview and some recommendations based on the computed data. We can list the following:

- A table displaying the percentage of good answers per question per assessment in the class (Fig. 3): this table, drawn with HTML, aims to summarize, for each question of a given assessment, the percentage of good answers out of the number of students having completed it. If this percentage is comprised between 0% and 33%, the corresponding field is automatically colored in red. The field is also colored in yellow (respectively in green), when the percentage is comprised between 33% and 66% (respectively between 66% and 100%).

- The pie chart giving the percentage of success per assessment (Fig. 4): we use the Highcharts library and the computed data to plot this chart which shows the percentage of good and bad attempts per assessments in the whole class out of the aggregate number of attempts done by the students.

- The histogram giving the percentage of success per assessment (Fig. 5): with the Highcharts library and the computed data, we plot this graph which shows for each assessment, the number of students in the class, the number of students having done the assessment, the number of students having failed, the number of students having succeeded in getting more than 50 points and more than 75 points.

- Some recommendations are also inferred from the computed data. For instance, if the percentage of good answers for some questions is less than 33%, we advice the teachers to reformulate them and better explain the related concepts in the videos. In addition, we inform the teachers when the mean of attempts is bad (equal to 0 or higher than 3) and we suggest some useful measures as it is shown by Fig. 6.

Thanks to this assessments statistics visualisation module, the teacher can know how his students are performing.

---

3https://www.highcharts.com/demo/bar-basic
Fig. 3. Percentage of good answers per question per assessment in the class

Fig. 4. Deep overview of the attempts per assessment

Fig. 5. Percentage of success per assessment

Fig. 6. Recommendations:

- **Pre-assessment**: It is compulsory to better explain in your videos the concepts covered in the questions 1, 3, 4 and to reformulate them. The mean of attempts before success is excellent.

- **Mid-assessment**: It is compulsory to better explain in your videos the concepts covered in the questions 1, 2, 3, 4, 5 and to reformulate them. The mean of attempts before success is too bad (still at 0 for the whole class).

- **Final assessment**: It is compulsory to better explain in your videos the concepts covered in the questions 1, 2, 3, 4, 5 and to reformulate them. The mean of attempts before success is too bad (still at 0 for the whole class).
V. YouTube Events

A. Introduction

In order to help teachers check if students are watching the videos and understanding them, we improve GCB by adding a data collector of YouTube events. The aim of these events is to know the behavior of the students while they are watching the videos, so that the system can infer suggestions and the teachers can improve the videos. The type of events that are stored are the following:

- Starting a video
- Paused
- Forward
- Rewind
- Ending a video

With these events, a teacher can infer information such as:

- If there are a lot of forward events in one part of the video, we can infer that students have the knowledge of that part. Thus, a teacher can remove that part of the video or add that part in an optional video.
- If a lot of students press rewind in one part of the video, it could mean that they are not understanding the concepts and they need to repeat that part. The teacher can add a new explanation or a new example so that the students will not need to rewind.
- If a lot of students pause the video at the same time, it could mean that the teacher is talking too fast.

B. Implementation

The YouTube events collector can be adapted to other platforms as it needs Javascript and AJAX in the client side, and a database in the server side. Therefore, it can be exported to other systems such as Khan Academy.

We implemented the collector of YouTube events using YouTube Player API Reference for iframe Embeds. This API allows embedding a video, controlling the player and notifying events with Javascript.

By default, this API notifies the following events: unstarted, ended, playing, paused, buffering and video cued. The forward and rewind events are not notified, so we had to implement them. We discover that every time one of the previous mentioned events happens, it sends two pause events one after the other: it is not consistent (after a pause event, there should be a playing event). Therefore, we use this sequence to identify forward/rewind event. In order to make a difference between a forward and a rewind event, we store the current time playing every second. When the sequence of two pauses is received, we compare the previous current time with the actual current time. If the current time is greater than the previous current time, then it is a forward event; if it is not, it is a rewind one.

After one of the previous events happens, a query is sent to the server with AJAX, containing the type of event (Event field), the ID of the video (Video field) and extra information (Info field) in JSON format, depending of the type of event.

In case of start and end events, the Info field is empty as it is not needed.

For pause events the Info field has two fields: time is the timestamp at which the video was paused, and duration is the number of seconds the video was paused. This event is sent after the video is played again.

For forward/rewind events, the Info field has: timeStart is the timestamp at which the video was playing, and timeEnd, the timestamp at which the student rewinded or forwarded.

Besides, a cookie is added so that the server can identify the user. The server stores in the Datastore the user identifier of the cookie, date of the query and the information of the event described above (type of event, video ID and extra information of the event).

VI. Dummy Data Generator

A. Overview

It is commonplace for present day Massive Online Open Courses (MOOCs) to provide service to a few thousands or even a few hundred thousands of students. In a significant number of occasions, the number of students greatly outgrows the expectations of the teachers and the developers of the platform being used to serve the class, resulting in unexpected behavior of the system. These problems could have been provisioned if each class was tested for its behavior under load.

There are surprisingly few tools that offer automated testing of MOOC classes though and as a consequence such problems come to the surface only after the class has gone online and a big number of students have enrolled. In this section, we present our approach to the automated testing of GCB. It should be noted that during the development of our automated testing tool, it became obvious that it could be used to assess not only the technical part of the development (generating the different possibilities and testing the correct behaviour), but also to simulate and test some pedagogical aspects as well.

We have not implemented the latter functionality, but given our existing work, it would be easy to extend it to generate data to match different personality types of students.

B. Functionality

In order to test the scalability, robustness, accuracy and performance of the learning analytics extensions in GCB analyzed in the sections above, we create an evaluation and testing tool called “dummy”. “dummy” is a python program which generates (fictitious) students, assessment data and YouTube events for populating the Datastore of GCB. This is achieved by generating low level HTTP requests that send the related data to the server (usually within a JSON formatted string). The data sent to the server are identical to the data that a real user would generate while accessing the course content. All the data are generated by the random number functions of python. At this point, “dummy” is tuned to work only with some of the assessments that are part of the GCB default course named “Power searching with Google”.

More specifically, it is able to provide answers to the three main assessments of that course (Pre, Mid and Fin). We
plan though to take advantage of the consistent format of assessments and activities created with GCB, in order to extend the functionality of “dummy”, to be able to provide dummy data for any assessment. At both the current implementation and in the case of generalization to any assessment, the following restrictions apply to each kind of question:

- Multiple choice (including true or false): no restriction.
- Multiple answer: no restriction.
- Numeric match: we send back to the server the correct answer without the final digit and the final digit is computed randomly. Thus the success ratio is $\frac{1}{10}$.
- String match: we send back to the server the correct answer without the final character and the final character is computed randomly. Thus the success ratio is $\frac{1}{26}$.
- Regular expression: These questions should be manually handled.

C. Implementation

In this section, we describe the technical details of “dummy”. In detail, “dummy” creates:

- **Students**: Different instances of students $s_i$ are being created and put in a vector $\vec{S}$. Thus, the fictitious class is the vector $\vec{S} = [s_1, s_2, \ldots, s_n]$. If the goal of the researcher is to test the technical aspects of the class, the only factor that differentiates the students is a serial number. All the answers that the fictitious students generate are created by a random variable that has a uniform probability distribution.

- **Assessment data**: After the vector $\vec{S}$ has been created, we can call the appropriate methods of each student instance to generate the related assessment data. To technically test a class, we pick a random answer for each question, then we calculate the score, and we finally send to the server a JSON formatted string with these data.

- **YouTube events data**: The key idea is the same as above. In order to test the technical aspect of the course, events are generated (along with their timestamps) for a specific video of the course according to a uniform probability distribution. Whenever an event is generated, it is sent at the server.

The program defines the class “Student()”. Every instance of that class is meant to represent a fictitious student. A unique number (AA) identifies the instances. The student class has the following methods:

- **Login**: it logs in the student instance to the server. This is a necessary step, since all the data that will be generated need to be associated with a student.
- **Register**: it creates the necessary record of the student to the GCB Datastore. As above, all the students should be registered.
- **PreAss**: it generates random answers to all the questions of the Pre Assessment of the class “Power searching with Google”, calculates the score and sends the results to the server in a JSON formatted string.

- **MidAss**: as above for the Middle Assessment of the class “Power searching with Google”.
- **FinAss**: as above for the Final Assessment of the class “Power searching with Google”.
- **YoutubeEvents**: it takes 2 integer arguments (the number of unit and the number of lesson). This method randomly generates any number (or none) of the following events (up to 10)
  - Forward
  - Rewind
  - Pause

In case the number of events is not 0, it also generates the following events at the beginning and the end of the video respectively
  - Started
  - Ended (in order to simulate students who do not finish watching a video, this event might not be generated, based on a certain probability)

The events are sent to the server as soon as they are generated, along with the related timestamps within a JSON formatted string.

- **Logout**: it logs out the student.

VII. CONCLUSION

Nowadays, there is an abundance of data available in learning platforms, but it is not always easy for teachers, students and other stakeholders to interpret them. This fact can often overwhelm the education professionals, instead of aiding them enhance the content of their course. There is a need of pre-processing these data, so that they are more accessible to teachers. To this end, we have presented a novel extension for one of the most prominent MOOC platforms currently available, Google Course Builder.

Our contribution is threefold: first, we enhance the above mentioned platform adding some first level information regarding the assessments available in the platform. More specifically, we aggregate student accomplishment data, generate plots and charts visualising their performance and attendance, and provide suggestions to the teachers, so that they can identify the weaknesses of their courses. Second, we capture data from the interaction of students with the YouTube videos embedded in the course. For this purpose, we utilized the YouTube API which notifies events related to changes in the player, such as “Pause”. Moreover, we improve this functionality adding events like “rewind” and “forward”. Finally, we implement a dummy data generating program in order to test the robustness of our approach when it has to handle a big number of students, as well as evaluate the accuracy of the recommendation system.

VIII. FUTURE WORK

We plan as future work to extend the assessment learning analytics module, by adding timer entries (generic timer of open tab and active tab timer, i.e. How much time the user is actually looking at a tab).

In the case of YouTube events, we intend to use the data that we collected to present different plots to the teachers and give
them recommendations to improve the videos of the course. Moreover, some additional statistics will be computed to give recommendations to the teachers.

Finally, we plan to explore (initially at a superficial level) the pedagogical potential of "dummy" and extend its functionality to generate data for any assessment created with GCB.

ACKNOWLEDGMENT

We wish to thank the Google Course Builder development team, and in particular Sean Lip, for useful suggestions. Google has no responsibility for the developed code.

REFERENCES


