From heterogeneous activiteis to unified analytics dashboards

Iván Martínez-Ortiz
Dept. of Software Eng. & Artificial Int.
Universidad Complutense de Madrid
Madrid, Spain
imartinez@fdi.ucm.es

Iván Pérez-Colado
Dept. of Software Eng. & Artificial Int.
Universidad Complutense de Madrid
Madrid, Spain
ivanjper@ucm.es

Dan Cristian Rotaru
Dept. of Software Eng. & Artificial Int.
Universidad Complutense de Madrid
Madrid, Spain
drotaru@ucm.es

Manuel Freire
Dept. of Software Eng. & Artificial Int.
Universidad Complutense de Madrid
Madrid, Spain
manuel.freire@fdi.ucm.es

Baltasar Fernández-Manjón
Dept. of Software Eng. & Artificial Int.
Universidad Complutense de Madrid
Madrid, Spain
balta@fdi.ucm.es

***Abstract*—Teachers often wish to integrate activities from disparate sources into their courses. For example, gamified activities, mediated through technology, can promote the type of active learning required to develop higher-level engagement by students. However, unless the activities have been designed to facilitate it, integrating their analytics into a single dashboard can require significant development effort. A general solution to such heterogeneous analytics integration can be of great value, by presenting a single view of student actions throughout the different parts of a course. We describe the problems presented when integrating the analytics of three heterogeneous stand-alone activities, in the context of a EU project to improve software engineering teaching. The idea is to increase student engagement via gamification, and explore the design space of possible solutions for providing integrated analytics over the heterogeneous activities. We then describe the design of a proof-of-concept implementation, based on the use of both xAPI trackers and simple CSV files for information exchange, single sign-on, a minimal class management web application, and updates to the analytics platform to allow dynamic changes in the multi-level analysis. The resulting approach can be readily applied to similar heterogeneous scenarios.**

Keywords—learning analytics, serious games, analytics

I. INTRODUCTION

Teachers have an increasing number of online activities that they can use in their classes, and are encouraged to do so by the pedagogical trend towards active learning, since interactive online activities can be much more motivating and engaging than traditional homework. For example, a gamified quiz can promote class involvement and provide valuable formative feedback to students and teachers alike. Since the use of these activities by students can generate insights into the learning process, it is desirable to offer teachers a single place where these insights, and the data that supports them, is collected. We term this a unified teacher analytics dashboard.

A typical analytics dashboard feeds from an analytics server, which in turn collects and analyzes individual student interactions with particular activities. The complexity of the setup varies according to the number of different activities to report in the dashboards, how they relate to each other, and the degree to which they are capable of interoperating with the analytics server. The simplest scenario is that of a single analytics-enabled activity, where the student (or the student's environment) would provide the activity with a unique identity ID (to allow the dashboard to distinguish between different students, and to tie together all interactions by a single student), and the activity would then periodically report student interactions to the server, which would validate the interactions, store them for future reference, and perform any necessary updates to the student's state as required by the different visualizations to be displayed by the dashboard. Typical visualizations include student-activity-over-time, student-degree-of-activity-completion, and student-degree-of-activity-success. Examples can be found in [1]. Note that, even in this simple scenario, teachers expect to be able to group students into groups, and possibly groups that have used the same activity to each other.

A more realistic scenario arises when multiple activities must be integrated together. If the activities are designed to work with the particular analytics system, they will be able to correctly report interactions with unique student IDs. Teachers could simply use multiple single-activity dashboards to check on student progress. However, this is unwieldy: teachers expect both class overview dashboards that integrate all class activities and per-student dashboards with all activities by a single student, instead of constantly switching between per-activity dashboards. Furthermore, activities are often grouped together in a hierarchy, which teachers expect to be replicated in the corresponding dashboards. If a course is structured into sections, teachers would expect aggregation to occur in the dashboards between activities of each, and overall for the whole course. This necessitates configuration to inform the analytics server and dashboard of activity grouping, and possibly relative weights and other aggregation-related details. An example of this integration can be found in [2].

A final complication emerges when the activity supporting tool was not originally designed with analytics in mind; or their implementation for analytics is devised to be exclusively consumed inside the tool and not by third-parties. In such cases, support for (limited) analytics may have to be added to each activity after the fact, while also obtaining a suitably unique student-id to include in each trace.

This paper focuses on the last two scenarios, which we term heterogeneous activities. The paper is structured as follows: Section II describes the initial problem encountered by an EU project seeking to improve student engagement via gamification, and explores the design alternatives for building a unified analytics dashboard for similar requirements.

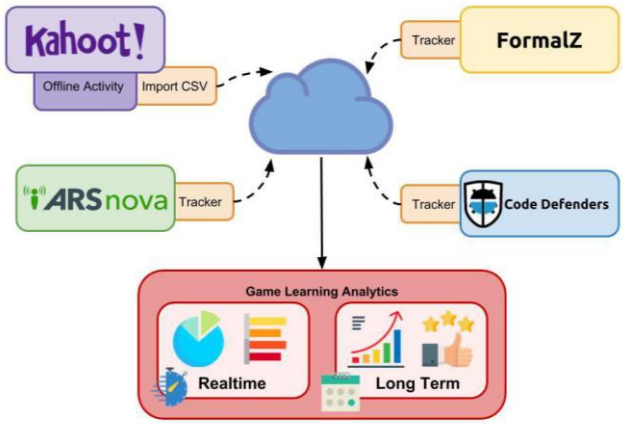

Fig. 1. Motivating example: a set of heterogenous activities, reporting user actions via trackers and, in the case of Kahoot!, CSV files to an analytics server.

Section III describes our approach, based on the use of the xAPI-SG vocabulary and minimal user configuration. Finally, Section IV discusses our preliminary results and comments on the generalizability of our results and possible lines for future research.

II. Case Study: EU Gamification Project

We motivate this paper by presenting the specific example of an EU project which would benefit from integrating heterogeneous activities into unified dashboards: IMPRESS.

The goal of IMPRESS is to improve the engagement of students learning software engineering via gamification, while emphasizing quality-assurance (QA) techniques. In software engineering, QA techniques are often relegated in favor of design and coding, which are perceived as less laborious and more creative. To achieve its goals, the project has selected several independently-developed, stand-alone online activities, none of them with support for the type of analytics that we would like to build a dashboard with, and each with completely independent authentication, class management, and basic analytics. The activities, depicted in Fig. 1, include:

- Interactive quizzes (via ARSNOVA.click), where students compete for quick and correct answers. The server-side component uses NodeJS.
- A formal-specification game (FormalZ), where students build specifications that correctly reject invalid inputs and, for valid inputs, correctly describe their corresponding outputs. Implemented with Java.
- A unit-test writing game (Code Defenders), where students compete to build tests that detect all bugs, and introduce bugs not detected by current tests. Implemented with Java.
- Generic results from offline activities (in this case, Kahoot! [3]), reported as CSV files with a simple structure.

Although access to the source code for 3 of the 4 activities is available (that is, for all excluding Kahoot!), our goal is to achieve analytics and dashboard integration while making only minimal changes. An additional requirement is for analytics and dashboards to be able to accommodate additional activities, while remaining flexible enough to allow activities to be added and restructured with minimal expertise and effort, even after the course has begun. In this sense, the last bullet point, referring to "offline activities", would allow teachers to conduct, say, a field visit to a museum, and later that day add the activity and upload a file describing student actions, which would later be available as any other type of activity-reported analytics.

III. Design and Proof of Concept

We are building on an open-source pre-existing analytics platform, built for the EU H2020 RAGE project. This platform [4], [5] uses the xAPI standard to communicate activity interactions with a learning analytics (LA) server, and provides libraries that facilitate analytics integration for both Java and JS-based activities. Since they facilitate tracking student activities, we call these components "trackers". The trackers handle authentication with the analytics server, as well as ongoing communication and, when necessary by poor network conditions, local caching and retransmission. Trackers also require the activities themselves to provide two credentials: the unique identifiers of both the user and the activity being performed. These are necessary to allow analytics to tie different sessions by a same user together, and to distinguish between different activities for a single student.

The remainder of this section describes each of the identified issues in greater depth. Subsection A deals with user-ID unification and single sign-on. Subsection B describes how teachers can string different activities together to build gamified courses. In subsection C, the focus is on offline activities. Subsection D dives into the internal changes necessary to accommodate dynamic activity trees, while subsection E describes how activities are informed of their context-of-use, and finally, in subsection F, we estimate the overall implementation effort for heterogeneous activity integration.

A. *Unifying User-IDs*

The first step towards unified analytics is to establish a set of common user-IDs to be reported by the trackers in each activity, so that traces belonging to the same player can be identified as such across activities.

Many institutions use learning management systems (LMSs) that can act as LTI [6] consumers; for this case, it would be advantageous to retrofit activities to act as LTI providers, which would then transmit enrollment and student-id to the activities at launch time. However, in this project, several of the participating institutions do not have institutional LMSs, so we discarded implementing an LTI-based solution.

Other options include the use of a single-sign-on (SSO) mechanism, such as Simple SAML, which is a popular open-source SSO implementation based on the Simple Authentication Markup Language [7]; or OAuth [8], which, while usable for SSO, is more popular for 3rd party authentication. For instance, OAuth is frequently used to login using existing social media credentials. There are many OAuth identity providers, such as Google or Facebook, and such 3rd parties can be avoided altogether by installing and hosting a self-owned provider. Regardless of the chosen variant, we consider the use of SSO highly important to ensure usability: the sense of immersion and enjoyment that can be provided by games would be broken if the player is constantly

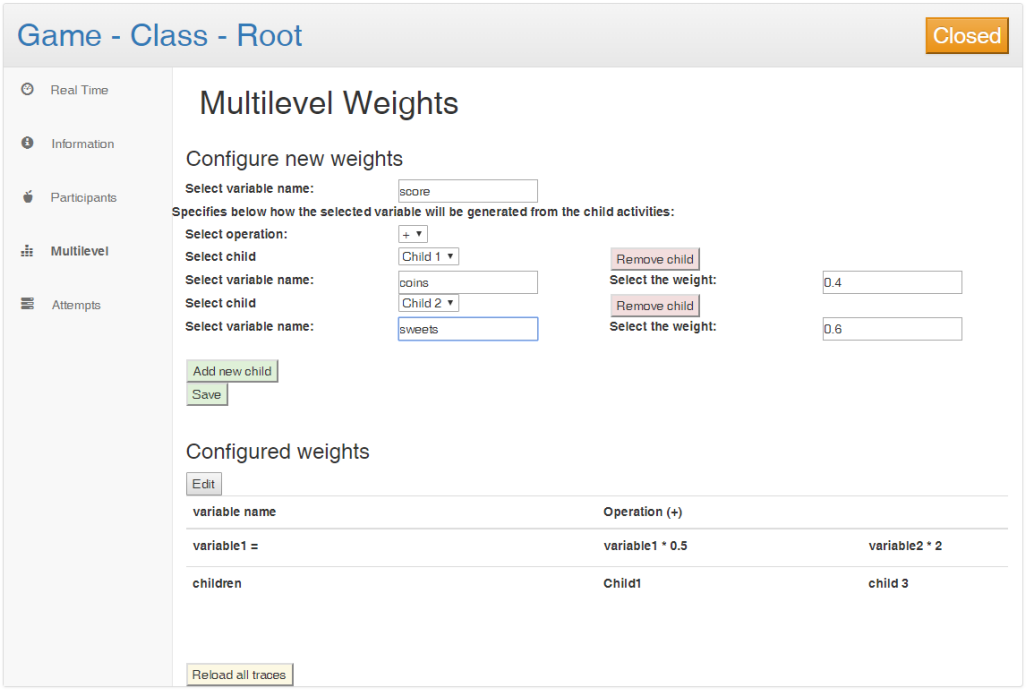

Fig. 2. Screenshot of the activity manager, displaying weights of two sub-activities. Clicking the "reload all traces" button (bottom left) will discard and reprocess all traces, using updated formulas.

forced to provide distinct logins and passwords when switching from one activity to another.

For the project described in this paper, we have opted for Simple SAML, due to its simplicity when compared to OAuth2; and the use of institutional e-mail addresses as unique, cross-activity identifiers for all players. This choice requires an SSO server per cluster of participating institutions, and changes to the activities to rely on this server instead of using their own authentication and class-building mechanisms. Each activity authenticates student-ids available to their tracker components by first communicating with the SSO server, which returns a token that can be recognized as valid with the analytics server.

In terms of changes to the activities, each activity has been modified to report, to its tracker, valid tokens, which are checked server-side to ensure that they correspond to existing users within the local SSO. To obtain the tokens, the activities have been modified to delegate authentication to their Simple SAML server.

B. Managing Activity Trees

The next requirement is to provide a mechanism that allows teachers to structure different activities into a larger activity, and later allow them to track player progress along these activity trees. Since all activities are web-based, it makes sense to implement this activity manager as a web application, using the same SSO mechanism described in the previous subsection.

When building dashboards for activities, teachers expect not only to be able toad and remove activities, and to add sub-activities to any given activity; they also expect to be able to vary the weights of sub-activities. For example, given a parent activity composed of a short 10-question interactive questionnaire, and an estimated 30-minute unit-test writing activity, we could wish to set their relative weights as 30% and 70%, respectively. We refer to the process of updating parent-activity progress and success as "activity rollup", mirroring the terminology used in IMS Simple Sequencing [9], and it involves 3 parts:

1. Input and storage of the formula and parameters to be used during rollup; together with the actual structure of the activity tree.
2. Evaluation of the formula, allowing the parent activities' progress and success to reflect those of its child activities
3. Display of results

In a previous EU project, BEACONING, which also supported hierarchical activities [2], the full activity trees, including rollup formulas and parameters, were authored and maintained outside the analytics system. In effect, step 1 was performed outside the system. Since, once authored, the formulas and parameters for an activity tree could never change, the analytics system adopted a very simple approach to step 2: a machine-readable description of the structure of the activity tree and its rollup formulas was available to the analytics system, which used it to update the success and completion of parent activities whenever these values changed in a leaf activity. Step 3 would then, for any specific dashboard, simply retrieve and display the latest values.

However, in this project, the analytics system no longer receives such a static tree. Instead, the activity manager allows teachers to dynamically modify the activity tree that they will use, and teachers can alter the structure and formulas in the tree at any time, even while students are working. This has necessitated important changes to the underlying analytics server, which are described in subsection C. The activity manager's interface, displayed in Fig. 2, simply provides the interface for step 1, hiding the underlying complexity from teachers.

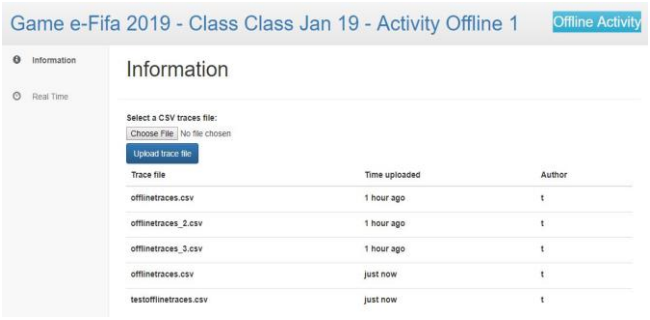

Fig. 3. Creating traces from a CSV file, allowing offline activities to generate similar analytics to online (tracker-enabled) activities.

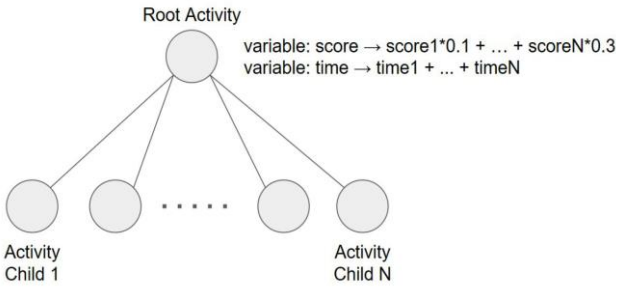

Fig. 4. Diagram of the rollup process, where root activities update their variables based on the values present in their child activities.

Note that, depending on the activities, it may make sense to track and update (via rollup) multiple sub-activity variables, which can then be visualized in the corresponding dashboards. The activity manager automatically creates and initializes a "score" variable, initialized so that all child activities contribute equally to the parent activity's score. This variable is being edited in the screenshot found in Fig. 2. Two additional rollup rules, for variables "variable1" and "variable2" are also present, but, in Fig. 2, they are not currently being edited.

To enable dynamic modification of activity trees, the activity manager provides a "reload all traces" button (visible in Fig. 2). Pressing this button forces all previous calculations to be flushed, temporally disrupting analytics visualization until they can be reprocessed with the new weights. While the new weights from previous traces are being evaluated, traces from activities can still be received, but they are added to a queue, and will not be processed until all older traces have been finished.

C. *Offline Activities*

Fig. 5 is a screenshot of the web interface that allows offline activities to generate similar traces to tracker-enabled activities. While this particular interface requires teacher intervention to upload the activity results, submitting the form actually calls a public and well-documented API in the server, which could just as easily be called directly from within a suitable activity. This enables certain types of tracker-less analytics integrations.

The CSV files to be sent must comply with an existing schema, where each line corresponds to an xAPI-SG [10] trace, and which is expected to use one of a small subset of valid xAPI-SG verbs, including *progressed*, *completed*, and *selected*. As a proof of concept, we have also built a Kahoot!-specific import interface that can load Kahoot! Excel result spreadsheets. This is a proof of concept for similar activities which already generate reports, although they do not follow our expected CSV format – it is relatively simple to write a small conversion tool that translates from the activities' internal format to that which is expected for our system's offline activities.

D. *Updating Activity Trees*

The analytics server maintains two main types of data-stores: one contains validated and authenticated interaction traces received by students, in xAPI-SG format; and another contains the results of analyzing these traces using the then-current analyses. Visualizations are built from the second of these two stores, which we will call the *results* store, as opposed to the *raw* store. Clicking on the "reload all traces" button from Fig. 2 results in discarding all potentially affected parts of the results store, to be recalculated using the updated formulas from the original data kept in the *raw* store. A third type of store is a distributed, fault-resistant *queue*, which receives incoming traces until they can be processed, is also used for rollup purposes (see Fig. 2), and into which the relevant raw traces are added when they need to be reprocessed due to a flush.

To implement activity rollup (see Fig. 4), all traces have a target activity. For incoming traces received from trackers, their target is simply the activity into which they are embedded. When an activity is first added to an activity tree using the activity manager, it is assigned a unique identifier, allowing the same activity to be used multiple times within a single tree without one use interfering with the others. The long hexadecimal strings listed as "children" in Fig. 2 contain the activity IDs of its children activities. Parent activities, although typically not backed by a true underlying activity, are also assigned activity ids; and these IDs are used to facilitate rollup as illustrated in Fig. 2: whenever variables change values in a child activity, a synthetic rollup trace is added to the queue, addressed to its parent activity, which, when processed, may result in cascaded updates to all ancestors of the original activity.

The rollup process may seem inefficient – after all, it would be faster to perform updates on all ancestors at the same time, instead of repeatedly enqueueing synthetic traces to be later processed. The advantage of our current streaming architecture is not so much in performance, but in scalability, robustness, and generality: by keeping individual trace processing simple and stateless, we can easily scale the number of workers threads doing the processing, while relying on the fail-safe nature of the distributed queue to ensure that, even if processing fails, no data will be lost, and the pending traces can simply be reprocessed at a later moment.

At any given time, the results store contains a snapshot of the state of all activities for all players. Incoming traces are quickly processed, updating the snapshot; and dashboards are built and updated from the contents of these results, leading to almost real-time visualizations. We are working on reducing the delays produced when the "reload all traces" button is pressed. Since traces are processed in the order in which they were originally received, dashboard results can then be assumed to be correct up to the time of the last correctly-processed trace – and incomplete afterwards, at least until reprocessing finishes.

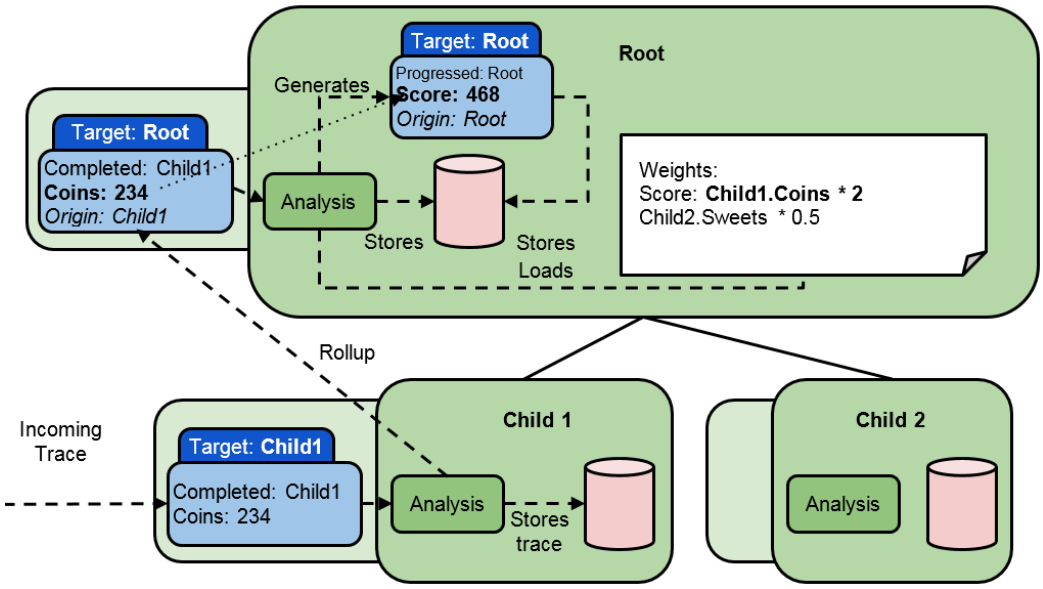

Fig. 5. Illustration of activity rollup. An update to the "Child1" activity, in the form of a trace received from a tracker embeded in the activity, causes two secondary rollup traces to be enqueued, which update variables in its parent ("Root") activity. A user-provided formula (white note within the Root activity) provides the formula to use. Each activity is annotated with a data-store, representing the fact that, conceptually, all activities keep their results up-to-date.

The strategy followed to allow calculation updates in a streaming architecture is termed *kappa* [11], in opposition to the more classical *lambda* architecture, where, in addition to a streaming analysis, a batch process is launched whenever large quantities of known data must be (re)analyzed. While batch processes are generally more performant than streaming for such a use-case, maintaining two separate but equivalent analyses, one for real-time updates and one for large recalculations, would have incurred in significant development and maintenance costs.

E. *Informing Activities of their Context of Use*

Different versions of an activity may appear in different parts of a given activity tree. For example, an interactive quiz on good unit-test strategies may be followed, after a few intervening activities, with another test on good formal-specification practices. In this case, each instance of the activity will have to report its own activity ID – one for the unit-testing quiz, and another for the formal-verification quiz. How does the quiz activity server know which activity-IDs to include in its traces? The general answer is that only the activity tree has this information, and therefore the simplest way to communicate these IDs to activity servers is to include them as URL parameters. So, as an example, given a unit-test quiz activity with 1234 as its activity-ID, and 5678 as the activity-ID for a formal-specification one, the following URLs would allow students to launch each, informing the activity server of the activity-ID that it should use when reporting interactions with either activity:

https://quiz.server/q?activityid=1234&quiz=unit-test
https://quiz.server/q?activityid=5678&quiz=formal-spec

These URLs require additional support on the activity-server side, to parse the activity-ID field and include it in all analytics traces generated from the activity. Given a base URL for an instance of an activity, the activity manager can easily create a version with an added "activityid" URL parameter. For example, given https://quiz.server/q?quiz=unit-test, and the ID "1234", we would have the first of the above examples. This parameter would then be ready to be decoded by the server before being included in future outgoing traces.

In the case of offline activities, there is no need to supply a context-of-use: the activity ID can be added when importing the CSV into traces.

F. *Estimating Implementation Effort*

The development effort required to allow dynamic modification of activity trees was a one-time task, as was the prototyping of the activity manager interface. Most of the effort required to provide integrated learning analytics to the heterogeneous activities in this case-study was related to the necessary changes in these activities to implement single-sign-on, and add correctly-configured trackers to the activities that could authenticate with our servers and report on student interactions with the activities.

In a best-case scenario, with clean, documented activity source-code readily available, we estimate that around 500 lines of additional code (without counting library code that may need to be imported in) will be needed to adapt a single activity to use a tracker; and a similar amount to write a custom internal-to-CSV converter for offline activities.

IV. Conclusions and Future Work

This paper addresses the integration of multiple heterogeneous activities, potentially including off-line activities, into a unified analytics dashboard to improve a software engineering course. We describe a working proof-of-concept integration and the underlying design decisions. While this is a particular scenario, we consider it to be easily generalizable to many other cases and integrations: any such integration will have to integrate tracker components into participating activities, or rely on after-the-fact uploading of their activity logs; configure those trackers with shared user-ids; configure the server to indicate who participates in each class, what the activity hierarchy looks like, and how it should be presented and aggregated for analytics purposes; and inform activities of their context so that they can generate

informative traces to be interpreted by the analytics server. The description of the changes to the analytics architecture to support dynamic changes to multi-level analysis configuration may also interest designers and users of such systems.

As future work, we intend to validate the activity manager and the overall integration of the component activities by performing several experiments with actual students from the participating institutions. The activity management interface itself is a working prototype, and multiple user interface improvements are still pending. For example, we will add a small reminder during trace reprocessing, to inform users of the timestamp of the last correctly-reprocessed trace; this will hopefully make the interface seem more responsive during recalculations.

Our work is available as open source at the E-UCM github repository (github.com/eucm), and builds on an existing open-source analytics platform and standards. Interested parties are invited to download, test and comment or even improve both the class manager and the activity tree evaluation.

ACKNOWLEDGMENT

This work has been partially funded by the Regional Government of Madrid (eMadrid P2018/TCS4307), by the Ministry of Education (TIN2017-89238-R) and by the European Commission (RAGE H2020-ICT-2014-1-644187, BEACONING H2020-ICT-2015-687676, Erasmus+ IMPRESS 2017-1-NL01-KA203-035259).

REFERENCES

[1] E. Duval, "Attention Please! Learning Analytics for Visualization and Recommendation Erik," in *Proceedings of the 1st International Conference on Learning Analytics and Knowledge - LAK '11*, 2011.

[2] I. Perez-Colado, C. Alonso-Fernandez, M. Freire, I. Martinez-Ortiz, and B. Fernandez-Manjon, "Game learning analytics is not informagic!," in *IEEE Global Engineering Education Conference, EDUCON*, 2018, vol. 2018–April.

[3] D. G. Perrin, E. Perrin, B. Muirhead, and M. Betz, "Kahoot! A digital game resource for learning," *Int. J. Instr. Technol. Distance Learn.*, 2015.

[4] C. Alonso-Fernandez, A. Calvo, M. Freire, I. Martinez-Ortiz, and B. Fernandez-Manjon, "Systematizing game learning analytics for serious games," in *IEEE Global Engineering Education Conference, EDUCON*, 2017.

[5] B. F.-M. Cristina Alonso-Fernandez, Ivan Perez-Colado, Manuel Freire, Iván Martínez-Ortiz, "Improving serious games analyzing learning analytics data: lessons learned," in *Games and Learning Alliance conference (GALA Conf), December 5-7, 2018, Palermo, Italy.*, 2018.

[6] J. Fontenla, R. Pérez, and M. Caeiro, "Using ims basic lti to integrate games in lmss—lessons from game× tel," *Glob. Eng. Educ.*, 2011.

[7] N. Ragouzis, J. Hughes, R. Philpott, E. Maler, P. Madsen, and T. Scavo, "Security Assertion Markup Language (SAML) V2.0 Technical Overview (OASIS)," 2007.

[8] D. Hardt, Ed., "RFC 6749: The OAuth 2.0 Authorization Framework," Oct. 2012.

[9] IMS Global, "IMS Simple Sequencing Information and Behavior Model, version 1.0 final specification," 2003.

[10] Á. Serrano-Laguna, I. Martínez-Ortiz, J. Haag, D. Regan, A. Johnson, and B. Fernández-Manjón, "Applying standards to systematize learning analytics in serious games," *Comput. Stand. Interfaces*, vol. 50, pp. 116–123, 2017.

[11] Nicolas Seyvet and I. M. Viela, "Applying the Kappa architecture in the telco industry," *Online article, May*, 2016. [Online]. Available: https://www.oreilly.com/ideas/applying-the-kappa-architecture-in-the-telco-industry. [Accessed: 14-Jan-2019].