GLAS: framework to improve assessment in educational videogames

Ángel Serrano Laguna

MÁSTER EN INVESTIGACIÓN EN INFORMÁTICA. FACULTAD DE INFORMÁTICA
UNIVERSIDAD COMPLUTESNE DE MADRID

Trabajo Fin Máster en Sistemas Inteligentes

Curso 2011/2012

Director:

Baltasar Fernández Manjón

Convocatoria: Junio
Calificación: Sobresaliente

Autorización de difusión

Ángel Serrano Laguna

21/06/2012

El/la abajo firmante, matriculado/a en el Máster en Investigación en Informática de la Facultad de Informática, autoriza a la Universidad Complutense de Madrid (UCM) a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a su autor el presente Trabajo Fin de Máster: "GLAS: framework to improve assessment in educational videogames", realizado durante el curso académico 2011-2012 bajo la dirección de Baltasar Fernández Manjón en el Departamento de Ingeniería del Software e Inteligencia Artificial, y a la Biblioteca de la UCM a depositarlo en el Archivo Institucional E-Prints Complutense con el objeto de incrementar la difusión, uso e impacto del trabajo en Internet y garantizar su preservación y acceso a largo plazo.

Resumen en castellano

El proceso de evaluación es fundamental para valorar los resultados de cualquier proceso educativo. Además es un elemento crucial en la aceptación de un nuevo proceso educativo por parte de los profesores. Esta evaluación se hace aún más complicada cuando se ha de realizar sobre juegos educativos. Learning Analytics es una nueva disciplina que aboga por colectar la información derivada de la interacción de los estudiantes con cualquier material educativo on-line, para por un lado entender mejor como interactúan los estudiantes con este tipo de material, y por otro, evaluar el resultado de los procesos de aprendizaje. La mayoría de aplicaciones de Learning Analytics que se están realizando en el campo de e-learning son sobre sistemas de gestión de enseñanza o Learning Management System. Los videojuegos educativos, debido a su alto nivel interactivo, presentan el entorno ideal para este tipo de procesos. En este trabajo se presenta la aplicación del proceso de Learning Analytics sobre videojuegos educativos. Este trabajo está organizado en tres partes: en la primera, se presenta la motivación científica del planteamiento de este proyecto, una propuesta de implementación abstracta y una propuesta de implementación concreta sobre el motor de juegos educativos eAdventure como objetivo; en la segunda parte, se detalla todo el proceso de implementación llevado a cabo en la realización del proyecto; y en la tercera y última se presentan algunas de las conclusiones obtenidas y se identifican las líneas de trabajo futuro a desarrollar.

Palabras clave

análisis de métricas de aprendizaje, videojuegos edcuativos, evaluación, minería de datos, eAdventure

Abstract

Assessment process is key to evaluate results of any educational process. Besides, it is a crucial element in the acceptance of any new assessment process by the teachers. This evaluation gets more complicated when it must be done over educational videogames. Learning Analytics is a new field that advocates of capturing all the data derived from interaction with on-line educational resources, first, to better understand how students interact with this type of resources, and second, to evaluate the educational action itself. Most of Learning Analytics research in e-learning is being deployed over Learning Management Systems. Educational videogames, due to their high interaction level, present the ideal environment for this type of analysis. In this project, application of the Learning Analytics process over educational videogames is presented. This document is organized in three parts: in part I, it is presented the scientific motivation for the project, an abstract implementation approach and a concrete implementation deployed over the eAdventure game engine; in part II, all implementation process is detailed, approaching all the project's subsystems; and finally, in part III, some conclusions of the project are presented, and some future work to be developed.

Keywords

learning analytics, educational videogames, assessment, data mining, eAdventure

Table of Contents

List of Figures

List of Tables

Part I

Motivation and Objectives

Chapter 1

Introduction

In traditional education, either in higher education or in other levels, the main evaluation method is based on written final exams [20]. This method, as some authors have pointed out [21], presents a different issues. These issues are related not only to the student evaluation, but also to the evaluation of the educational activity itself: the amount of data available is limited, and it is usually restricted to students and educators subjective perceptions (e.g. through polls about the past courses). Other metrics, mostly based on exam grades, might not give enough information about the educational activity, or whether it was a success or a failure and why. Moreover, these data usually become available when the activity is finished or when it is too late to make an intervention, improvement or correction in the ongoing action.

With the emergence of the Web, on-line educational resources have grown exponentially. Many institutions now use LMS (Learning Management System) to organize their courses, to allow students to communicate among themselves and with teachers, and to improve access to educational resources [27]. Still, despite all of these on-line resources, evaluation is still usually performed using traditional methods. Most of the content presented in LMS is finally evaluated through written exams in classrooms, or through online tests or exams.

However, there is a whole new body of data, derived from the student interaction with on-line educational resources. These data can be collected and analysed not only to improve the evaluation methods, but also to obtain real-time feedback about the progress of any

educational activity, enabling educators to predict results and react to that progress.

Data mining processes are used in multiples disciplines to analyse big amounts of data looking for knowledge patterns (discovery of regularities or rules) for further use. For instance, this type of processes are used by companies trying to improve their services analysing their client profiles and offering them only those products that might interest them, and by scientists looking for solutions for data categorization problems or ontology creations in fields like bioinformatics or medicine.

In the past few years, a new discipline focused in the data mining over learning metrics, and generically named as *Learning Analytics* [22], is trying to determine how this type of processes can be used in the data analysis to improve any of the many aspect of the educative process.

Essentially, Learning Analytics is the measurement, collection, analysis and reporting of data about learners and their contexts, for purposes of understanding and optimizing learning and environments in which it occurs [22].

These ideas have been successfully applied in other disciplines, like Business Intelligence, a well-extended set of techniques for analysing business data to support better business decision-making [28], or Web Analytics, where internet data are collected in order to understand and optimize web usage [23].

Learning Analytics results can be applied at different levels and with diverse purposes: at course-level, where teachers and students obtain a better perspective of the educational process and its results; at course-aggregated level, where predictive models and success/failure patterns can be found in the analysed data; at administration-level, where more detailed statistics can be obtain for groups of students or schools in order to, for example, improve resources allocation; and at regional/state level, where all results obtained in all schools or faculties can be compared.

Until now, the available set of parameters to analyse educational outcome was not enough to establish the conditions of success or failure in the different educational processes. Nowa-

days, mainly due to new mobile technologies expansion as well as behaviour change, students interact with a big amount of educational resources, from multiples places and platforms, producing traces that can be collected and analysed. Therefore, the amount of available data has increased significantly. This fact, coupled with the evolution and improvement of analysis techniques as well as with the current processing power of computers, favours the establishment of Learning Analytics Systems which help to improve the educational system as a whole.

Learning Analytics process is divided in several steps: first, data selection and data capture, which is established depending on the subsequent analysis goals; second, data storage in an appropriate datastore; third, data filtering and structuring; fourth, integration with other relevant data acquired from another sources; fifth, data analysis and knowledge extraction; sixth, representation and visualization of that information; and seventh and final, taking actions with some effect over the global system. And then, all steps begins again, resulting in an iterative process.

Horizon Report 2011 marks Learning Analytics as one of the new technologies to adopt in the next four or five years [26], denoting the relevance of this field in the future. This idea is reinforced by the fact that Learning Analytics is one of the lines of research and funding contemplated by the Seventh Frame Program (FP7) of the European Union.

Despite of being a relatively new discipline, some research about Learning Analytics have been accomplished by several researchers, producing solutions as LOCO-Analyst or SNAPP, which will be detailed in chapter 2. Most of this research is focused in the interactions performed by the students with LMS, whose contents are mainly based on static resources with low user interactivity, as documents, presentations or videos. However, there are others educational resources, as simulations or games, which present a bigger interactivity level and, therefore, generate a higher and more varied amount of data to be analysed.

For example, in the last few years, Game Analytics is being used to let developers know about how players interact with their games. One of their main purposes is to identify where

and why a player got stuck during the game, so game developers can try to smooth this hardness, to avoid player frustration and thus keep him engaged and playing [25].

The proven educational advantages of simulations and games [18], like the active involvement of the students or the increased motivation, is promoting that teachers, from different areas and levels, start to use this type of activities to teach their lessons and also to make their students develop skills with a new approach. New situations and challenges, related with learned skills and far from written exams or tests, can be presented to students in games and simulations.

However, most of these activities do not have an important weight on the final student evaluation, because most of games and simulations lacks of an appropriate assessment system able to generate rigorous and reliable student results. And even when this assessment system exists, it is used with auto-evaluation purposes only, and it is unable to collect and export that information for subsequence analysis.

Aware of these problems and needs, the final goal of this project is to apply data mining process and learning analytics in games and simulations to obtain a general model that allows their optimization and improvement. This goal can be split in several sub-goals:

- Establish which generic data must be captured for an effective evaluation of educational simulations and games, and which of these data depends on the individual characteristics of each simulation and game (field, scope, interaction mode, deployment device) and which data have a more general nature.

- Obtain analysis models able to provide reliable and concrete assessment results from the captured data, as well as models focused on spotting games and simulations weakness, which educational aspects do not cover, or in which ones results are not as expected, enabling an iterative improvement of the games and simulations.

- Study the best and most adequate visualization methods to present this type of information.

- Stablish adaptation rules, based on the data, that allow to provide with adapted experiences for each of the students.

To cover all these goals, a prototype framework will be developed, using eAdventure educational platform as game engine to execute the games and simulations, where all user data interaction will be generated.

Chapter 2

Previous work

Learning Analytics is a relatively new field. There is a lot of theoretical work, but when it comes to find software implementing its principles it is hard to find available and complete solutions that approach all Learning Analytics steps. Specifically, at the time of writing, we were not able to find any publicly available system that applies the Learning Analytics ideas in educational games, although there is a growing discipline around games and parallel to Learning Analytics, Games Analytics, that is being used by games companies to understand how their customers play their games.

Below, some of the main software implementing Learning Analytics is presented.

2.1 LOCO-Analyst

"LOCO-Analyst is an educational tool aimed at providing teachers with feedback on the relevant aspects of the learning process taking place in a web-based learning environment, and thus helps them improve the content and the structure of their web-based courses" [8] (Figure 2.1).

The generation of feedback in LOCO-Analyst is based on analysis of the user tracking data in a Learning Management System. These analyses are based on the notion of Learning Object Context, represented by a student or a group of students, interacting with a learning content: reading, quizzing, chating.

It uses Semantic Web technologies to annotate, with the facilities of the *Knowledge and*

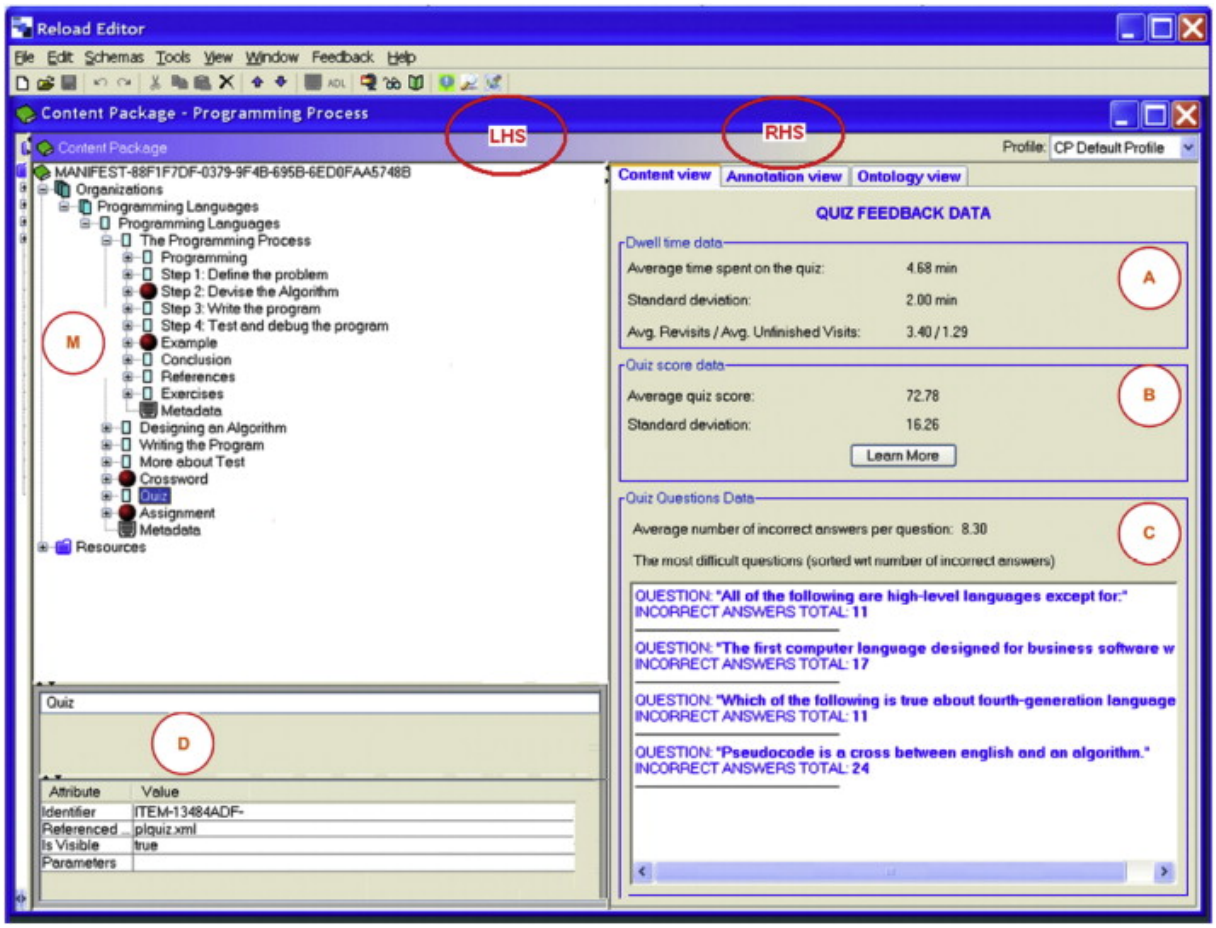

Figure 2.1: *LOCO-Analyst screen capture*

Information Management (KIM) platform [8], learning artifacts. They define the LOCO (Learning Object Context Ontologies) ontological framework, to annotate and interrelate these artifacts that comprehend lessons, tests or messages exchanged during online interactions.

Finally, LOCO-Analyst is implemented as an extension of the Reload Content Packaging Editor [10], an open-source tool for creating courses compliant with the IMS Content Packaging specification.

2.2 SNAPP

SNAPP (Social Network Adapting Pedagogical Practice) [15] is a software tool that allows users to visualize the network of interactions resulting from discussion forum posts and replies (Figure 2.2).

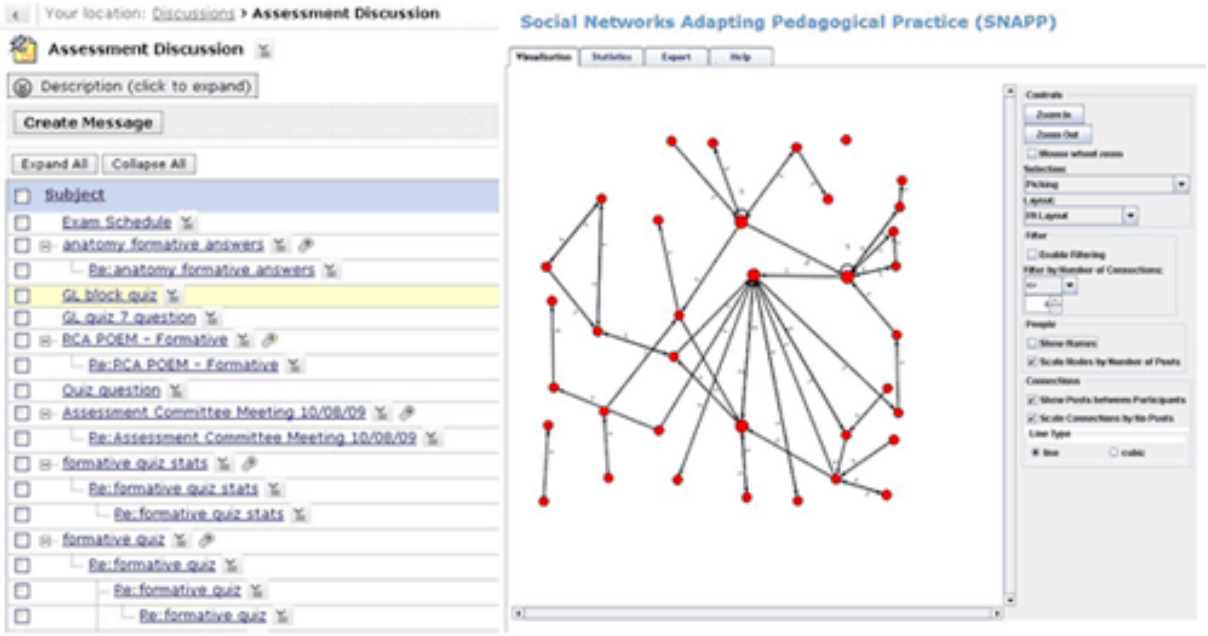

Figure 2.2: *SNAPP tool screen capture*

As LOCO-Analyst, it is usually deployed in Learning Management System, and it uses students interaction data in forums (who posted replies, who began a discussion...) to create interaction diagram as seen in the figure 2.3.

This diagram can tell several things: identify disconnected students, identify key information brokers within the class, identify potentially low performing students so interventions can be planned...

2.3 Conclusions

Learning Analytics is on the rise. It is beginning to be used mainly in e-learning and in combination with LMS. It is also known that similar approaches have been used in

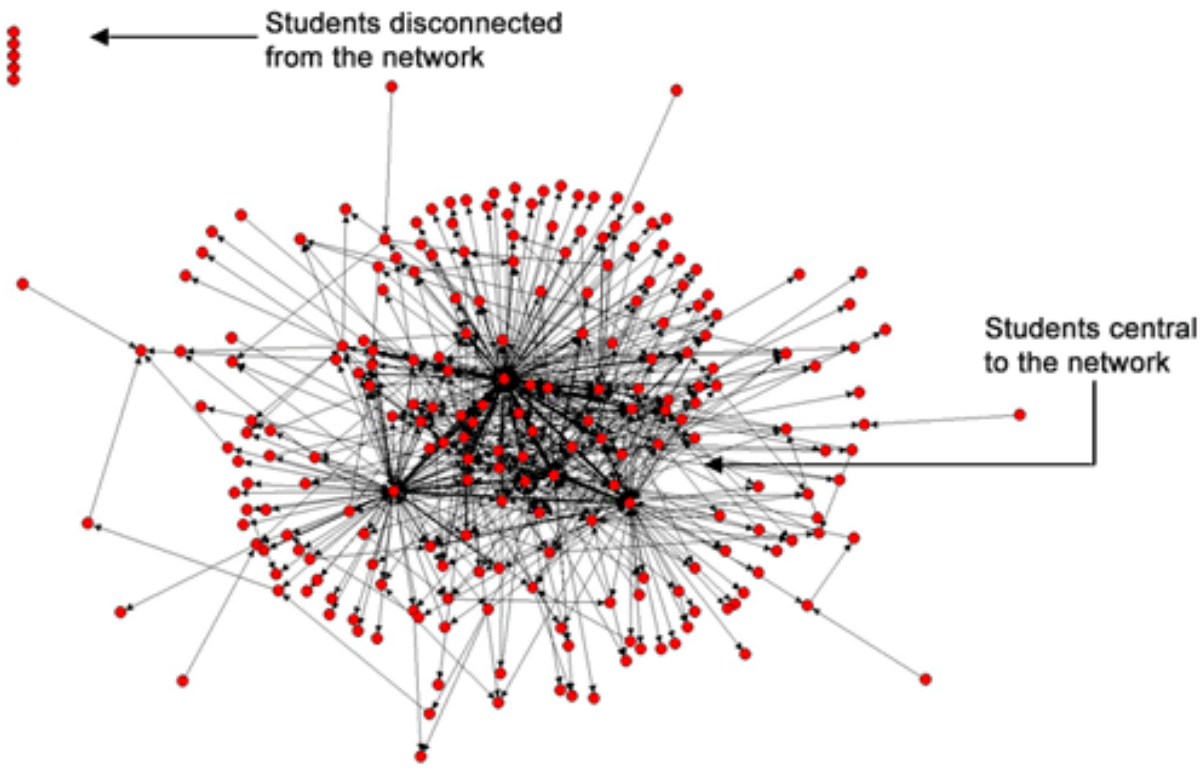

Figure 2.3: *SNAPP diagram showing students interaction*

commercial games to identify users difficulties and adapting games to the users so they play for more time (also known as keeping the user in *flow* state [19]). But for now there are not publicly available frameworks or even models to apply Learning Analytics to games.

In the next chapter, a first theoretical approach of a Learning Analytics Systems focused on educational games is presented, using as a guide the Learning Analytics Steps.

Chapter 3

Learning Analytics in educational videogames: first approach

Authors agree that Learning Analytics process [21] begins with selecting the most relevant data to be captured. These data can be generated by students or other users. Once it is captured, data must be aggregated and transformed into information (for example, using charts or other visual representations). With this information, the educator should be able to judge how the student used the educational resource. Some authors call this step predict, since information is converted into knowledge, and knowledge enables predictions. However, in our approach we will use this step to assess the student, and for now on, we will name this step as assess. Assessment information can be used, under certain conditions, to dynamically assist the student, and to refine the educational resource, based on the students results. Finally, all the knowledge acquired can be shared with others whom could benefit from it. Table 3.1 represents a scheme with all the steps and their descriptions.

Below, we discuss the main steps in the Learning Analytics process, and how these steps can be particularized in educational games, proposing a theoretical Learning Analytic Model and a Learning Analytic System.

To support all these steps, we propose a system based on a Learning Analytics Model (LAM) that defines all the information required for every step, and a Learning Analytics System (LAS) that comprises all the processing power required by the model.

Step	Unit produced	Description
Select	Data	Choose the data to be captured
Capture	Data	Collect selected data
Aggregate and Report	Information	Sort out captured data and convert it in information
Assess	Knowledge	Understand reported information and convert it into knowledge used to assess students
Use	Knowledge	Adapt the system based on assessment
Refine	Knowledge	Improve educational action
Share	Knowledge	Show knowledge for the benefit of others

Table 3.1: *Learning Analytics steps*

In this approach, focused on educational games, we consider the LAS as a separate system that is in continuous communication with the game engine. The LAS also has access to the game model and the LAM (Fig. 3.1). The LAM is defined by a set of models, which are directly related to the different modules contained by the LAS (Fig. 3.2).

In this chapter the learning analytics steps are described in general and for educational games in particular, building the LAM and the LAS upon them. Available data in games are the starting point to define the LAM. These data are used as starting point and all steps are built from them

3.1 Select and capture

First, the data to be captured by the LAS are selected. These data is the raw material that feeds the subsequent steps. The data selection criteria is driven by the educational resource objectives. Also some constrictions such as technical limitations and privacy policies must be taken into account. In educational resources, meaningful data can be selected from personal information about the student (e.g. age, gender, etc.), academic information and any other data provided by the resource context.

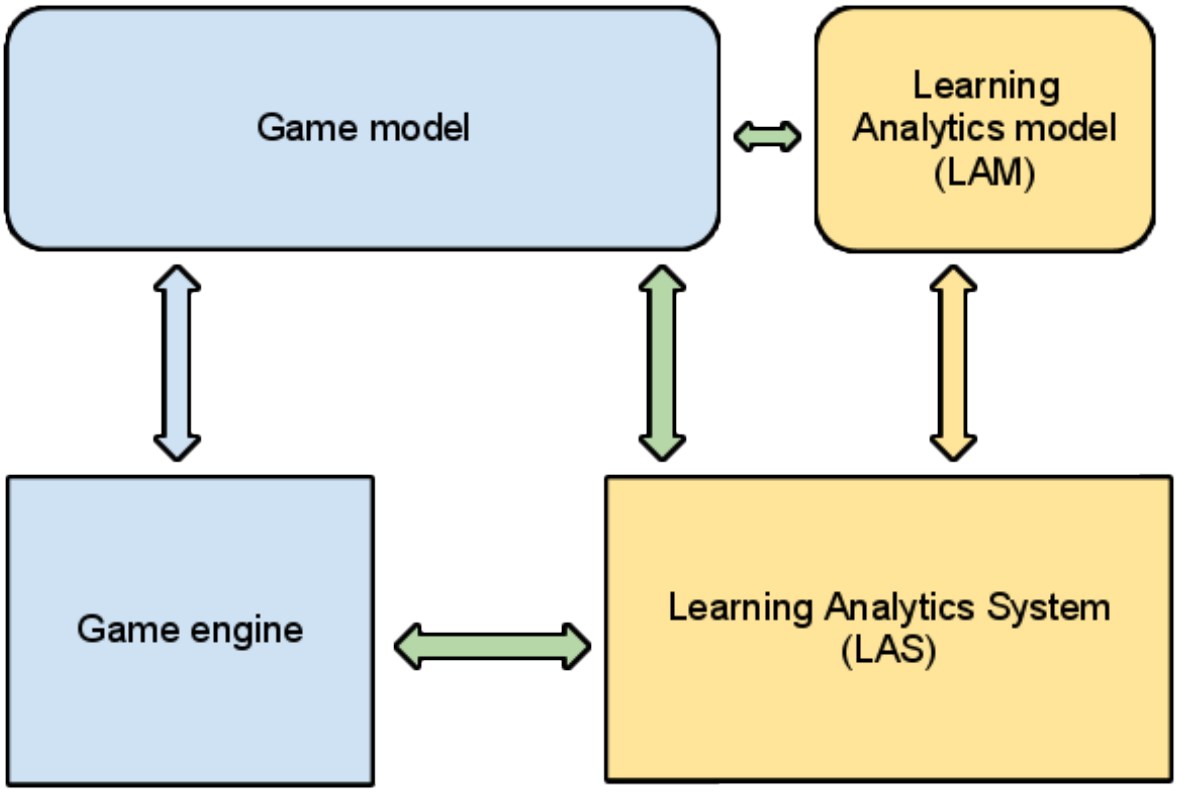

Figure 3.1: *Relation between the different components involved in the Learning Analytics process. The LAM is dependent on the game model and the LAS. LAS is aware of the LAM and the game model, and can communicate with the game engine.*

While in static resources (e.g. PDF files, videos, presentations...), the only data that can be obtained are the number of views and the time spent with them, the interactive nature of educational games provides a whole new type of data that can be selected:

- **GUI events performed by the student during the game**: mouse clicks, keys pressed, and other events (joystick movements, buttons interactions), depending on the input method. Not only the event itself can be recorded, but also the time when it occurred and whether it was performed over a target (e.g. some click over a game object). These events can provide clues about the student behavior during the game (e.g. if all GUI events were captured the LAS would be able to recreate the complete game play).

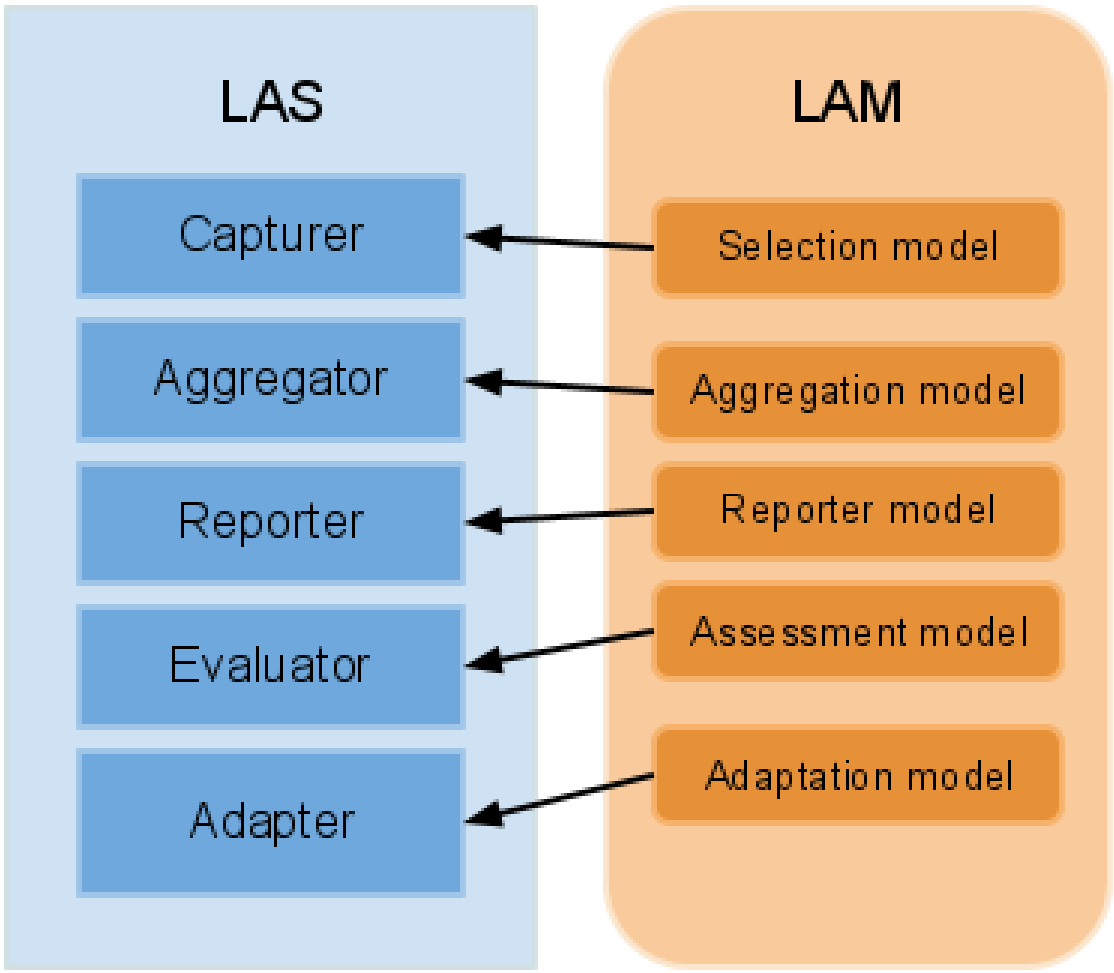

Figure 3.2: *The LAS consists of a series modules that take part in the different steps of the Learning Analytics process. The LAM holds models defining the information required for every step. The LAS uses the LAM to process all the data captured and generated.*

- **Game state evolution**: the game state is represented by a set of variables and their values that specify a concrete status in the game instance. The evolution of variables during the game execution represents the game state and the achievement of the game goals. Depending on the case, the whole game state evolution could be recorded, or it could be recorded only at some points (e.g. when a phase ends, or a goal is achieved).

- **Logic events**: a logic event is anything that moves forward the game-flow. Changing

the value of a variable, finishing a phase, launching a cut-scene (i.e. a slide-show or video), losing a life, achieving a defined goal, etc. Some logic events, and their timestamps, can be directly related to the student progress in the game, and thus be relevant for the assessment.

Selectable data are limited by the technologies used to deploy the games: Not all the data proposed here is available in every game platform. These data are platform-dependent and must be defined in the LAM's selection model. All games must define its own model according to their own purposes.

Once the data are selected, the framework requires a way to capture that data. The technology involved will be very important to establish how the data are collected. Access to different internal parts of the game engine is required to capture some of the information. This implies, for instance, that such model cannot be generally applied to commercial games provided only as executable files and then behaving as black boxes.

Another issue is the moment when the captured data are communicated to the LAS to begin the processing. The simplest way is to store all the data locally and send it back to the LAS when the game is finished. Data could also be sent asynchronously in certain significant moments, like when the student ends a phase or achieves a goal. Other option is that all data is sent to the LAS as they are being captured. This last two approaches enable real-time assessment that can be used to help the student during the game. Depending on the needs, all these data might go through some kind of filter, for instance, to anonymize or encrypt the data.

3.2 Aggregate and Report

The captured data must be processed and organized to obtain significant information in an human readable formats, like tables or graphics. A more meaningful report can be done if the LAM contains, in the aggregation model, semantic rules to interpret all the received data. For example, the system could relate a raw event (e.g. a variable taking a particular

value) with a semantic event in the game (e.g. the player completed a goal). These semantic rules can be based on the game engine, where some events can have an implicit meaning (e.g. an engine where pressing escape key always brings up the menu) or on the game itself (e.g. if the game variable "hits" is "8", the phase is completed).

Semantic rules can be expressed like conditions producing new data to be reported: when a condition (based on GUI events, a logic event or a concrete game state) is met a new unit of data, defined in the aggregation model, is generated. For example, when in the game state, the variable *score* equals to 10, and the variable *gold* equals to 15, the LAS aggregator produces a logic event "Goal 1 completed". The reporter could then treat this event as any other logic event.

The LAS' aggregator needs to be endued with mechanisms capable of understating and processing these kinds of rules (Fig. 3.3).

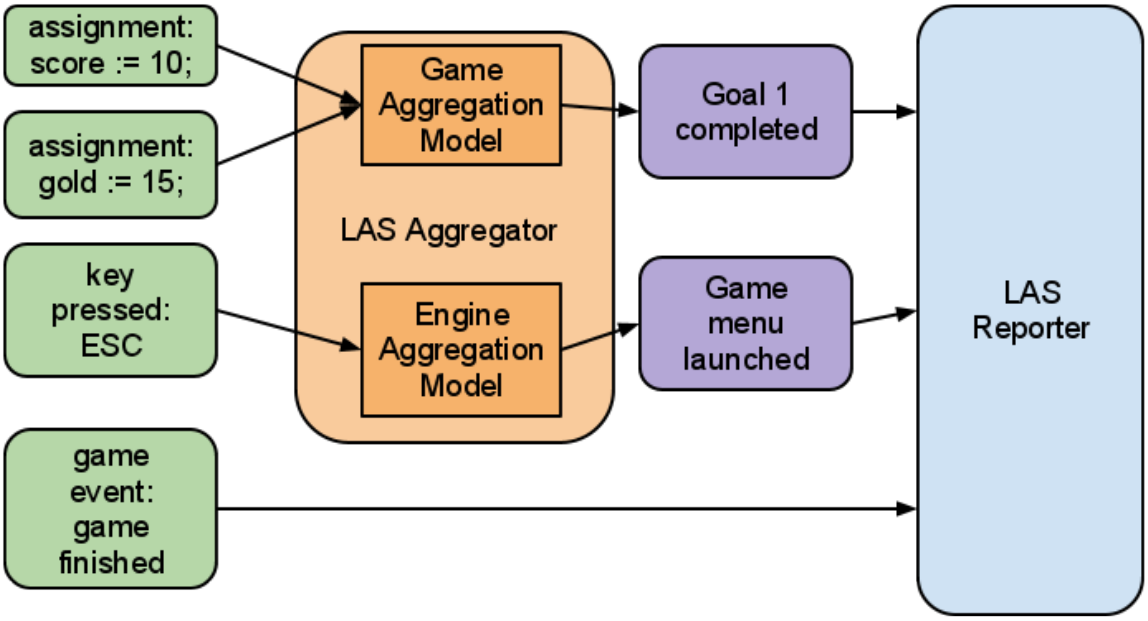

Figure 3.3: *Raw events are passed through the semantic rules contained by the LAM, and converted to more meaningful data. Events can be grouped and simplified through semantic rules. Some events, such as the game finishing, are considered as meaningful and do not need to be interpreted by any rule*

After aggregation, information can be reported in common ways, such as tables or charts, but also, it is possible to take advantage from the inherent characteristics of games to report information with new representations. For example, "heat maps" could be created for every phase, in which the heat can measure the amount of times the player clicked in every point of the phase, or the places where the player was defeated. If there is enough information about the user interaction, an animation recreating how the student played a game phase could be shown.

The reporter model contains which information must be reported and which representations must be used. Common reports can be defined at engine level (e.g. heat maps for every phase can be common for all games), as well as reports at game level, holding important information in that particular game.

These reports can be even richer if data from different students are aggregated. Average results can spot which goals took more time or the places where most of the students failed.

3.3 Assess and Use

The information and the reports generated until now can give an overview of how the students are using an educational resource. However, this information should have some practical consequences to be really useful. It is the moment to transform the received information into knowledge, and to use this knowledge to assess the student.

Games are organized around goals. In educational games, these goals should be based on the success in some educational aspects. In our context, and based on the concepts of the selection and aggregation process, we could have several types of goals, represented by:

- A GUI event or a series of GUI events performed by the student, over a game object or in total.
- A concrete game state, fulfilled fully or partially.
- A variable taking a defined value.

- The launch of a particular logic event.

These classes of goals are platform-dependant, and should be defined by the LAM's assessment model.

Compound goals can be defined based on these simple goals. An educational game can define all the necessary goals to cover all the educational aspects that are to be learned by players of the game. Based on these goals and with the reported information the game can be used to assess the student.

This assessment can have two applications: a) just to measure the success of the student in the game, and act accordingly (e.g. enabling the student to access to a new educational resources); or b), if the captured data are being passed to the LAS during game time, dynamic adaptation through real-time assessment (e.g. if the student got stuck in some point, the system will offer him help). Rules for this assistance are contained by the adaptation model, and are processed by the adapter, which is able to communicate with the game engine to perform the adaptation.

Assessment and dynamic adaptation could be more sophisticated. As some authors pointed out [10], propagating information through complex structures (e.g. Bayesian networks), can help to determine what is going on in virtual simulations, and better decide what adaptation profile to choose.

However, our approach pretends to be based on simple principles that can be understood by the average educator. Complex structures, like Bayesian networks, are normally out of reach for most educators.

3.4 Refine and share

With all the accumulated knowledge from previous steps, an educator can know about the global results (assessed in the previous step) of the educational action and can identify which educational goals were not achieved as expected. Thus, educators can refine the educational resources to improve the results or readjust their expectations.

In educational games, those game goals that were not obtained as expected can be detected. Aggregated data from several students can ease this job, pointing out, in average, which goals were more difficult to students. From here, the game could be modified or even redesigned to facilitate its accomplishment. This does not directly imply making the game simpler or reducing the educational goals, it could be enough to smooth the learning curve in the game, or adding some extra help in game states or situations that are specially hard for the users. Maybe, learning analytics conclusions could show that the student did not get stuck, but stop playing the game after a while, indicating that it was not engaging enough.

Finally, the LAS can share all the knowledge obtained with other systems. These systems cover from LMS to institution administrative systems. Making the data public can be an option in some cases. In order to be able to share data, some considerations such as privacy policies, what knowledge is shared or which standards are used in the communication, need to be taken into account.

Chapter 4

Implementation proposal: eAdventure

eAdventure is an educational game platform developed and maintained by the <e-UCM> research group at the Complutense University of Madrid for the last 5 years. This platform includes a game engine and an easy-to-use editor, targeted at educators. eAdventure is currently undergoing the development of the 2.0 version, where new features are being added. Some of these include support for multiple platforms [11] and an easy to use narrative representation of games [12]. Moreover, we propose to implement the framework presented in this paper in this new version of the system.

eAdventure games are composed of scenes, which can represent from a simple scenario in an adventure game where the player's avatar moves to a more complex slide-show, going through an array of mini-games and other content. These scenes are always composed of simpler parts referred to as scene elements, each of which will usually have a graphical representation, a position in the screen, behaviors, etc. The current scene and the status of elements in the screen are defined as the game state. It is the flow from one scene to another, behaviors of the scene elements and effects (changing current scene, showing text, launching videos, assigning values to variables) that make up a game, by continually changing the game state until a final state is reached.

The LAS is implemented on a server. It is initialized with the Game Model, (containing the Adventure LAM) and the Engine LAM. The LAS has several modules to satisfy the requirements for every step of the Learning Analytics process (Fig. 4.1). The relation

between the modules and the steps is detailed below.

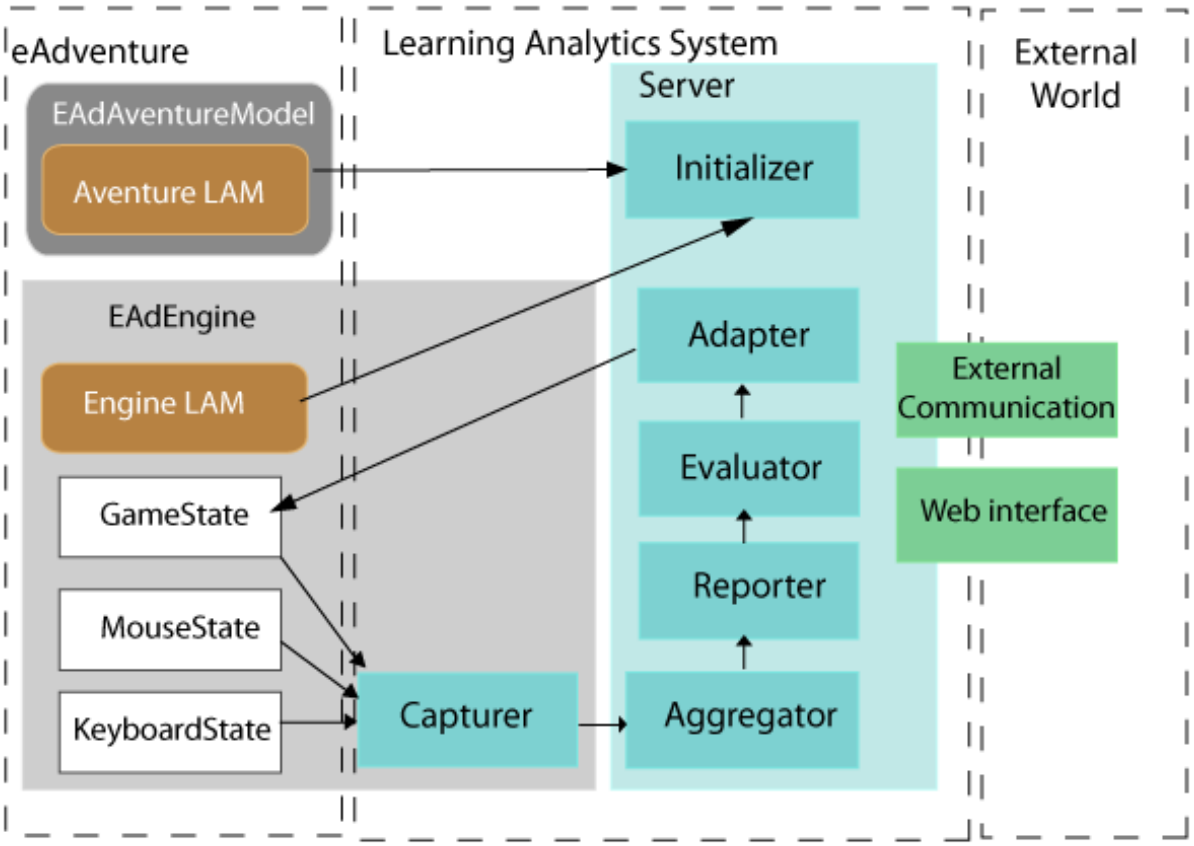

Figure 4.1: *General organization for the LAS integrated with eAdventure. The LAS is deployed in a server, and can communicate with the game engine. A web interface and external communication with other systems is offered as well.*

4.1 Select

Given that eAdventure is intended to be a general game engine, including its own editor, our proposal tries to make selectable the biggest amount of data to allow the game designer choose between all the available options. The eAdventure Learning Analytics Model define three units of selectable data:

- LAGUIEvent: represents a detailed GUI interaction. It holds the GUI event (mouse action, drag and drop, keyboard action) with its properties (mouse button, key pressed) and the target scene element, if exists.

- LALogicEvent: represents the launching of a game effect. It holds the generic effect data and additional information about the concrete instantiation (e.g., in the changing scene effect, the initial scene and the final scene).

- LAGameState: represents a game state in a certain moment. It contains a map holding all the game variables associated with their current value.

Every of these units has associated a timestamp, representing the moment the event occurred since the game was started.

In the editor, where and when data must be selected. For example, we can mark which type of GUI events (mouse, keyboard) we want to capture for every scene element (LAGUIEvent), and there is a special option to record all the GUI interactions performed in the game, allowing the recreation of the whole game play. It is possible also to capture those game effects that have relevance in the game flow (LALogicEvent) and, if desired, produce a LAGameState with the current game state. LAGameStates can be configured to be automatically generated periodically, when a game condition is met or when the game ends.

All these options are added to the selection model as part of the game's LAM, which will be used by the LAS' data capturer.

4.2 Capture

To capture all these data it is required a data capturer with access to all the relevant parts of the eAdventure game engine. The game engine has three main elements that are involved in this process: the input listener, which processes all GUI input from the user; the game state, which stores all the values for all the variables in the game at any time;

and the effect handler, which processes all in-game events (e.g. scene changes). All these elements communicate any relevant change to the data capturer, which then captures this information according to the current game selection model. The captured data are instances of the selectable data units presented before.

All these collected data are sent to the LAS aggregator (Fig. 4.2). Due to the multi-platform nature of the eAdventure game engine, different implementations take care of the communication with the aggregator. The data capturer can be configured to send out the captured data when the game is finished, a scene change happens, a defined condition is met or in real time.

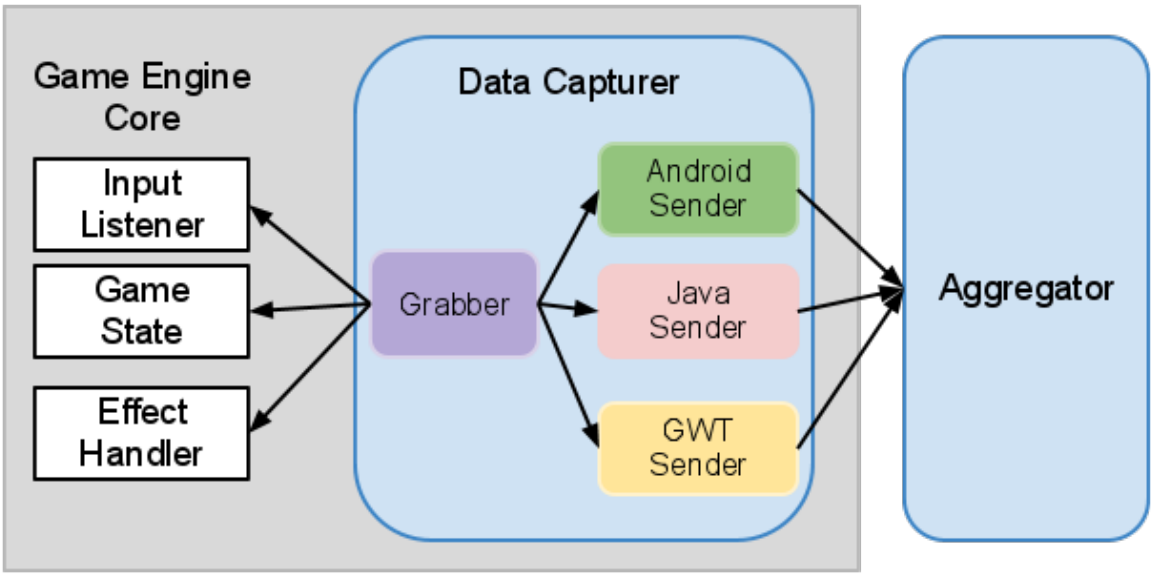

Figure 4.2: *The data capturer is compound by two elements: a grabber connected with all the elements producing significant events in the game engine, and a sender managing communication with the aggregator*

4.3 Aggregate

The data sent are received by the aggregator, who makes a first data processing based on the semantic rules defined by the aggregation model, contained by the Adventure LAM,

converting the basic units into more semantic pieces of data. This new units can be defined through the editor, as well as the rules of conversion from basic units (presented in the II.c section).

Aggregator also groups all the GUI events by type and scene element, filters redundancies and stores all the data in the LAS database.

Previous versions of eAdventure provided a basic mechanism for data aggregation and report generation. This basic mechanism allowed for information to be written in a textual report based on the values of the variables in the game. This way, the game author could define a set of rules that would, for instance, write in the report that the player failed to complete a goal if a variable to indicate this was given a certain value [17].

The same sort of data aggregation can be performed with the new LAS system. The rules in this case use a syntax that establishes the meaning of the data that were captured during the game to generate a report. The most basic reports will only include basic textual information, such as "Goal X was completed at time Y". However, more complex information can be aggregated to generate detailed information about the goal, such as "Goal X was completed by time Y, after Z attempts where the player failed to solve problems A, B, C, etc."

4.4 Report

The reporter represents in a web format all the information stored in the database. Among many others, the reports can be:

- A table relating scenes with the total time spent in any of them.
- Heat-maps showing where the player is hovering with the mouse most (Fig. 4.3).
- Screen capture recreated from game states.
- Game animations built from captured GUI events.

- Graphics showing the evolution along the time of selected variables.
- Tables with direct queries to the database.

Figure 4.3: *Heat map showing the concentration of left mouse clicks in a scene. Main heat zones are situated in interactive elements of the scene.*

All these data can be shown for an individual student or for groups of students. The system can be extended to add new reports generated from the stored data.

4.5 Assess

As established by the theoretical approach, this step is when the student is assessed. The assessment model contains all the goals established for the game. Goals can be defined

in the editor as variables taking certain values at given times. The evaluator takes these goals and checks them against the stored information.

The accomplishment of these goals can be viewed through the reporter, and can be sent to the adapter to enable dynamic adaptation or to be shared with external systems.

4.6 Use

When the data are being captured in real time, dynamic adaptation can be used in the game. Adaptation rules are defined in the adaptation model. These rules can be defined in the editor and contain:

- An effect, which is considered the adaptation event and could be any eAdventure game effect (showing a text, changing a variable's value, launching a video...)

- A condition, establishing when the effect should be launched. Conditions can be the general conditions offered to create game logic in eAdventure games, or conditions based on goal accomplishment (e.g. a goal is not completed when the time for doing it expires).

The adapter takes the current game state and the goals information offered by the evaluator to check adaptation model conditions. When a condition is met, it communicates to the game engine the effect to be launched.

4.7 Refine

To support the refine task the LAS offers, through the web LAS reporter, information about the individual goals. This allows, for instance, to identify those goals with worst performance by the students. How this performance must be improved is up to the game designer.

However, to ease this task, results obtained for the games' goals can be compared over time (checking if results improved after student played several times the game) and between different versions of the game.

4.8 Share

Nowadays, the selection and adhesion to standards for the content interoperability is an essential matter in the development of e-Learning contents. Current e-learning standards, like SCORM [14], are not designed to communicate all the information collected by our LAS with other systems. For this reason, the best way of taking advantage of the full potential of our approach is to develop specific ad-hoc communication solutions for the systems that take into account all these data (e.g. a Moodle plugin). This idea has been implemented to communicate an eAdventure activity inside a educational design in LAMS [13], where all the information can be gathered and shown to educators, and use them to modify the lesson flow in an automatic or monitored way.

In the near future, it will be feasible to implement our ideas in compliance with next generation of e-learning standards. For example, one the last initiatives leaded by the IMS Global Consortium, the IMS Learning Tool for Interoperability (IMS LTI), goes in that direction. This specification allows for the execution of learning tools hosted in external servers. Until other promising standards mature [15], our LAS is able to export all the information contained in the database along with the LAM required to interpret it into a exchangeable XML-based format.

Part II

Implementation

Chapter 5

Introduction

In this part, all the details of the currents LAS implementation are presented. For now on, the framework will be named as GLAS (Games Learning Analytics System). A general overview of GLAS is shown in figure 5.1.

The whole process begins in the eAdventure game engine, where all games are executed. User interaction is tracked by the GLAS Tracker, and after being filtered by the GLAS Selector, is sent to the GLAS Server, using REST calls. Data is stored in the GLAS Database. Finally, GLAS Reporter access stored data through the REST API to generate the reports.

It must be noticed that the theoretical model has been simplified in order to fit in the time limitation of this project. Rest of the approach is presented as future work in chapter 11.

Below, all the features expected for every sub-system are presented. The technologies considered to implement them are presented.

5.0.1 GLAS Tracker

GLAS Tracker is the link between the eAdventure game engine and the GLAS Server. Therefore, eAdventure should be able to communicate with both of them.

eAdventure is a multi-platform game engine (it supports Java, Android and HTML5), and the GLAS Tracker must be able to work with any of the supported technologies, pro-

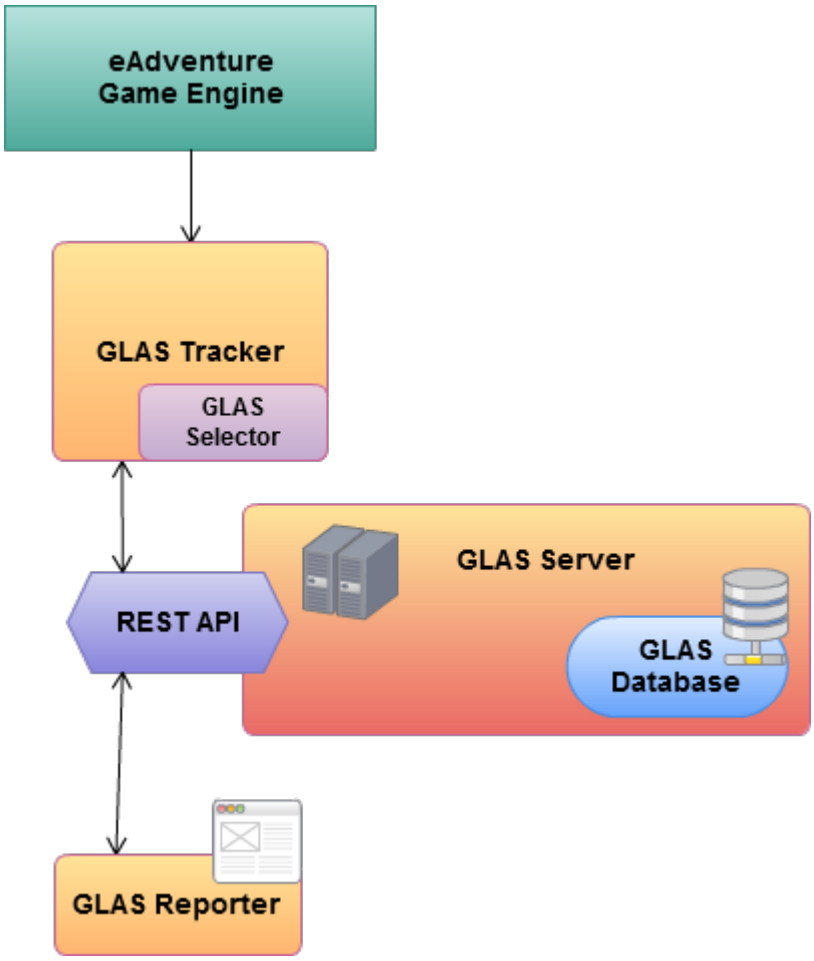

Figure 5.1: *Overview of the Game Learning Analytics System. The game engine sends collected data to the GLAS Server, using a REST API. This same API is used by the GLAS reporter to generate the reports.*

viding a general API for all platforms, and concrete implementations where needed.

GLAS Tracker must be able to store temporary data, to maintain the collected traces until they are sent back to the server. When possible, GLAS Tracker should run in a different process than the game engine, to avoid game freezing.

Also, GLAS Tracker contains a subsystem, the GLAS Selector, that must be able to filter data coming from the game engine. This subsystem must be configurable in order to enable different filtering settings for each game.

Finally, GLAS Tracker must communicate with the server, mainly to send collected

traces to it. eAdventure multi-platform nature must be taken into account since network communications are totally dependant on the implementing technology.

5.0.2 GLAS Server

GLAS Server must address two different needs: first, it must be able to receive and store data coming form the GLAS Tracker, and second, it must be able to serve these data, in the appropriate format, to the GLAS Reporter.

One of the most common solutions for this type of problem is the definition of a REST API that allows to receive and to provide data. For the implementation of this REST API, an ideal solution would be use a reliable library in the implementing technology that ease development and deployment, and that provides with multiple formats for the data communication.

GLAS Server should also include a solid and efficient database technology that allows for multiple concurrent accesses, and a secure access to the data.

Finally, GLAS Server must also have processes to register and identify users, to pre-process and post-process data as needed, and to easily communicate with the database.

5.0.3 GLAS Reporter

GLAS Reporter will access to the data contained by the GLAS Server, and it will generate reports based on the received data. These reports will contain statistical data about the game and they will be presented to different types of users: game designers, teachers, students...

Thus, GLAS Reporter must be able to communicate with the GLAS Server through the network, and it must be able to interpret and process data received from it.

Finally, it should have tools (charts library, graphics manipulation...) to represent the data in the appropriate format and appearance.

Chapter 6

Considered Technologies

As presented in chapter 4, several technologies have to be used in order to build the proposed framework. We have to consider three subsystems: three in the client side and one in the server side. The clients include the game engine (eAdventure) where the educational games will be played and where the track data will be captured (GLAS Tracker), and the reports system (GLAS Reporter), where analysis results will be shown to users. In the server side, first we need a server technology able to implement a REST API, and second, a datastore to keep all traces collected in the game engine. This datastore also will be used in order to generate graphics and data visualization in the reports system.

Next section presents the main technologies considered for the implementation are presented in the following order: server technologies, datastore solutions, tools for the reporter system and finally eAdventure as selected game engine.

6.1 Server technologies

The server side technology is used for two different tasks. First, it must deal with the datastore, implementing all the usual operations for it (insert, delete, update, read). And second, it must offer a REST API that allows for sending the data to the datastore. This API also must allow queries that return formatted data used by the GLAS Reporter.

6.1.1 PHP

PHP is the most extended server side language [14] and from its version 5, it is object-oriented. Most of hosting providers offers PHP support at low cost, it counts with a good documentation and it also has multiples features to connect with many types of databases. It also has several libraries to deploy REST services, like Tonic [16].

However, its interpreted nature makes it slower compared to other options (e.g. Java) and its lack of multi-threading functionality could make harder the implementation of some algorithms. Reuse and modularity can be hard to accomplish and, for big projects, a certain level of previous experience is required.

6.1.2 Java

Java is an object-oriented programming language that is used in desktop applications as well as in servers, executed in containers like Apache Tomcat. It supports multi-threading, it has features to connect with the most common databases and several libraries to create REST services, as Restlet [12] or Jersey [5].

On the other hand, most of the pre-built frameworks based on Java, like GlassFish, are too big and complex for the intended purposes of this project, and a new solution should be programmed from scratch to meet the system requirements. Also, in Java the level of complexity of system deployment increases.

6.2 Storage

In order to store the collected data, a datastore must be used. This datastore will be integrated in the server and it will address two aspects: enable storage operations (insert, update and delete data) and enable query operations. Two datastores have been considered:

6.2.1 Appengine DataStore

Appengine is a Google's platform intended for the easy deployment of web applications. It covers server side with Java and Python, and client side, if any, with the Google Web Toolkit. Among other services, the Google Appengine offers its own datastore to store and access data. Its main advantage is that it automatically handles concurrence and scalability issues, thus, if the datastore is attacked for an unexpected amount of traffic is able to adapt itself to it, without any downfall.

The drawbacks are that free Google's datastore plans has a limited number of transactions with the datastore, and this number is not enough for the expected amount of operations for the planned system, so a premium plan should be bought, with the additional costs associated. Finally, developers necessarily need to use the server technologies implemented by the Appengine (Java and Python).

6.2.2 MySQL

MySQL is the world's most used open source relational database management system [3]. It has complete documentation and most of host providers offers it as its main database system. It also can be easily combined with most server side technologies.

On the other hand, MySQL does not support the foreign keys that could determine relevant relation between some of the collected data.

6.3 Client Side: Reports System

In the client side, apart from the game engine capturing the traces, which will be presented in the next section, it is required some software able to represent the reports generated for the data to the different users. The chosen technology should be able to communicate with the REST server and to provide a graphic reports system enabling the generation of graphics and visual representations.

Considered technologies are presented below:

6.3.1 GWT

The Google Web Toolkit (GWT) is aimed to develop browser web applications in Java. When the code is compiled, it is translated into Javascript. GWT allows for the use of native Javascript code and thus all of the many charts libraries written for this language. It also includes AJAX features, enabling REST queries.

As drawbacks, Javascript is single-threaded, its performance is low and compatibility with new native functions (overall, the new functions added in the HTML5 specification) is irregular across browsers.

6.3.2 Java

Java counts with many REST libraries, as Restlet and Jersey, and there are many chart libraries, as JFreeChart [6]. However, developing a client in Java will imply installing an application to access the reports.

6.4 Game Engine: eAdventure

The eAdventure game engine [24] has been chosen as the game engine for deploying the games and capturing the produced traces. The eAdventure platform is open source, and in its version 2, is a multi-platform game engine able to execute 2D games in Android, Java and HTML5.

Some features must be added to the eAdventure code in order to enable the trace capturing.

6.5 Selected technologies

For the GLAS Server, Java will be used due to its better support in concurrence and its multi-thread features. MySQL will be used as database, due to being free of any cost associated and its level of maturity.

The GLAS Reports will be developed with GWT. Although it can be slower, most of new applications are web applications, and a report system accessible from a web browser is a better solution for the final user.

GLAS Tracker will be implemented in Java, Android and HTML5 (using GWT), to meet the multiple platforms supported by the eAdventure game engine.

Chapter 7

Generating and selecting traces

As raw material for subsequent analysis, traces collected from the educational game are needed. These traces must represent the meaningful events occurred during game play. Furthermore, these traces must be selected and filtered, in order to avoid storing irrelevant data in the database, or creating unnecessary overload in the tracking system. In this chapter, the traces' types in GLAS are defined. Available tools to select and filter these traces are presented. Finally, eAdventure game events must be transformed into GLAS traces, so a model and a implementation to perform this transformation is detailed.

7.1 Traces definition

In section 4.1, three types of traces were defined and considered for further analysis: GUI events (all data derived from the direct user input), logic events (representing meaningful events triggered during the game play), and game state events (information about all the variables and other data representing the game state). Although eAdventure is a game engine primarily conceived for a very specific game genre (point and click adventures), the traces defined in the GLAS framework try to be as generic as possible to simplify integration with other game engines.

In actual implementation, traces should be small and should contain the least number of fields as possible, and those fields should only contain simple data types (integers, floats and strings), because large amounts of traces are going to be sent through the network.

This idea creates a conflict with the third type of trace, that in which the game state was periodically sent. For two reasons: first, trace's size would be too big, since it supposes to contain all the game state data; and second, it was complex to clearly establish what information was actually relevant and valid for all the games.

Due to these problems, this type of trace was not included in the final implementation. Instead, logic traces contains all relevant information about the game state evolution. Valuable information is obtained from changes in the game, and it is difficult to get it from a game state that could contain hundreds of variables that only change once or twice in the whole game play.

Thus, two types of trace were defined. An UML diagram, representing the traces classes, is shown in figure 7.1. As seen in the figure, both types extends an abstract class, containing several common fields:

- **id**: an unique id defined for every trace. This id is usually initialized to null, and it is filled when the trace is stored in the server database. This field is mainly used for database purposes.

- **gameId**: Sometimes, GLAS framework will be deployed for several games. All these games will be sending their collected traces to the same server and database. This identifier establish the game which generated the trace.

- **userId**: A lot of users can play the same game reporting traces to the same server. This field contains the trace's owner identifier. This id is a reference to an unique player identifier, known by the server, which is associated to more information about the player, also stored in server's database.

- **userSession**: If it is established for the game, any user can play the same game more than once. This user session is a counter pointing to which of those plays the trace is associated. It is used to distinguish several game plays from the same user.

- **timeStamp**: It marks the amount of milliseconds since the game started until the trace was generated. From this field, it can be established the order in which traces were generated.

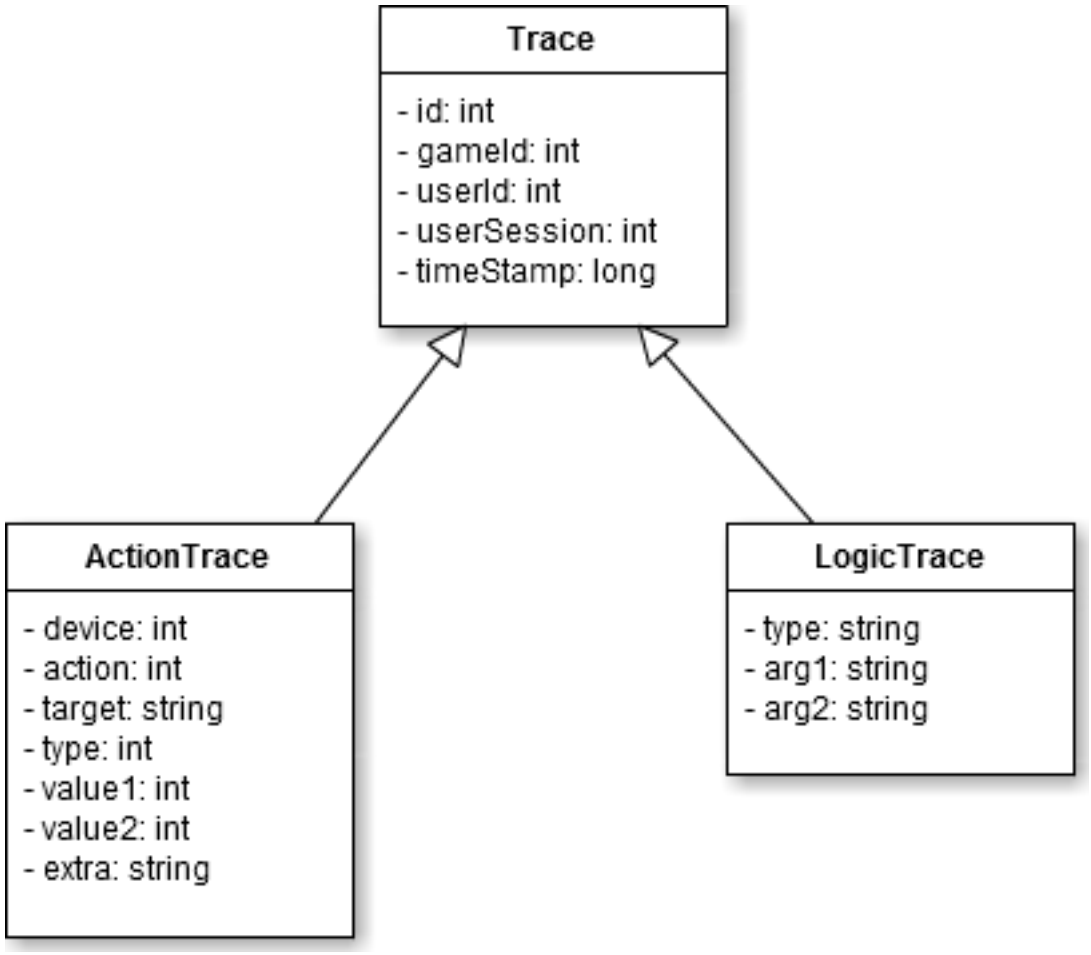

Figure 7.1: *UML diagram representing the two types of traces defined by GLAS. ActionTrace represents traces derived from direct user's interaction with input devices (mouse, keyboard, touch screen). LogicTrace represents those relevant events occurred during game play (phase change, a significant variable changing its value...)*

The two types of traces are constructed upon this generic trace:

7.1.1 ActionTrace

ActionTrace traces reflect the direct user's interaction with input devices. It contains the following fields:

- **device**: every device contemplated by the GLAS framework is defined with a different numerical constant. This field contains the constant of the input device that produced the trace.

- **action**: the concrete action that generated the trace. Several constants define every of the possible actions, constructed upon three fundamental actions: pressed, released and moved. Depending on the input device originating the trace, every of these actions has a different meaning. For example, in a mouse, *pressed* means that some button was clicked, but for a touch screen, it means that user pressed some point in the screen, or for a keyboard, that some key was pressed.

- **target**: it is a string with an unique identifier pointing which game element, if any, received the action event. For example, if some game object identified as "help button" was clicked, the value for target field in the generated trace would be "help button". This field might not be always generated, but when present it can be used to sort and classify traces.

- **type**: it contains the concrete type of the event, and again, its interpretation is dependant on the device which generated the trace. For example, in mouse events, this field contains the clicked mouse button (left button, right button or center button), but for keyboard events, contains an integer representing which key was pressed.

- **value1, value2**: these two parameters are integers and contain additional values for the trace. Those values that have no place in previous fields. For example, for mouse events or screen touch events, *value1* and *value2* respectively contain x and y screen coordinates, marking where the event was performed.

- **extra**: A final string field containing other relevant information. For example, in keyboard events, this field is used to register if some extra keys (like CTRL, ALT or SHIFT) were pressed when the event was triggered.

7.1.2 LogicTrace

LogicTrace traces contain data about the relevant events occurred during the game play that are triggered by the user's input actions: phase changes, significant variables updates, starting, ending or quitting the game, etc. As the nature of these traces can be diverse, depending exclusively on the game engine capacities, three generic fields has been defined in order to included the largest number of possibilities:

- **type**: a string defining trace's type, allowing events orderings and classifications.
- **arg1, arg2**: two attributes designed to store the event's arguments. Values for this fields will depend exclusively on the trace type. For example, for a change variable value event, *arg1* will store variable's id and *arg2* will store the new value assigned to the variable. In change phase events, *arg1* will contain an id marking the current scene and *arg2* the id of the next scene.

7.2 Trace selection

Not every generated trace is relevant for further analysis, hence, GLAS adds a trace selector, the GLAS Selector, programmed to decide which traces are relevant. Criteria for the relevance are defined in a configuration file, created by those interested in perform the analysis.

The two selection methods defined in the GLAS Selector can be observed in figure 7.2: one method is used to filter *LogicTrace* and another is used to filter *ActionTrace*.

As said before, criteria used by the GLAS Selector is defined in two separates files, one for every type of trace. File structure is the same for both types: Every line starts with a field name, followed by an equals sign and then, separated by commas, a set of values for that field. All traces arriving to the GLAS Selector and containing any of the fields specified in any of the lines with some of the given values, passes the filter and is selected.

For example, the following configuration file for *ActionTrace*

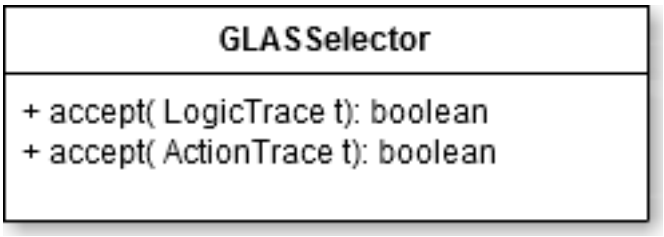

Figure 7.2: *UML diagram for the GLAS Selector. It has two methods used to decide which traces are selected and which are not.*

```
device=0,1
action=pressed
```

defines that only those traces coming for devices 0 or 1, or with action equals to "pressed", are selected.

Similarly, the following file can be defined for *LogicTrace* selection:

```
type=changeScene,changeField
```

defining that only those traces with value "changeScene" or "changeField" in the field "type" will be selected.

This model is extensible and, n the future, will be extended in order to support more complex combinations for the selection.

7.3 Producing traces in eAdventure

Once defined the traces used by GLAS, next step is to generate these traces in the game engine. In this case, in the eAdventure Game Engine.

First step is identify the correspondence between traces defined by GLAS and the elements that are internally used by the game engine. In the eAdventure case, the following relationships have been established:

7.3.1 InputAction as ActionTrace

eAdventure is a multi-platform game engine. It handles all the input events in the same way for every platform. A generic class, InputHandler, receives all natives events (generated from Java, Android or JavaScript) and converts them into generic eAdventure input events, named InputAction. Once this object is generated, it is passed to the game loop, which is responsible for determining what game element receives the input and for firing all the consequences programmed for that event (game effects).

InputAction contains all the necessary data about the input event. An UML diagram of this class can be observed in figure 7.3.

As shown in the figure, the correspondence between *ActionTrace* and *InputAction* is one-to-one: All fields defined in *ActionTrace* can be filled with the data in *InputAction*. Thus, a simple translation can be run over *InputAction* to obtain *ActionTrace*.

As said before, InputHandler seems a perfect place to intercept all user input interaction and convert it into *ActionTrace*. This integration will be detailed in chapter 8.

7.3.2 Effects as LogicTrace

eAdventure game play establishes its game flow over game effects. In eAdventure, game effects move forward the game. Effects can change variable's values, play animations, move game characters, change scenes, show the options menu, reproduce sounds, quit the game, etc. Everything that happens in an eAdventure game is produced by an effect, and these effects are usually fired by the user input interaction.

Every time an effect is launched, an EffectGO object is generated by the GameState. This object, as shown in figure 7.4, contains all the information about the effect that defines it. This effect can be of many types.

This event type is directly linked to those events represented by *LogicTrace*. Unlike *InputAction*, *EffectGO* have not got a direct translation into *LogicTrace*. Type field in *LogicTrace* can be a reference to the effect class, however, *arg1* and *arg2* fields must contain

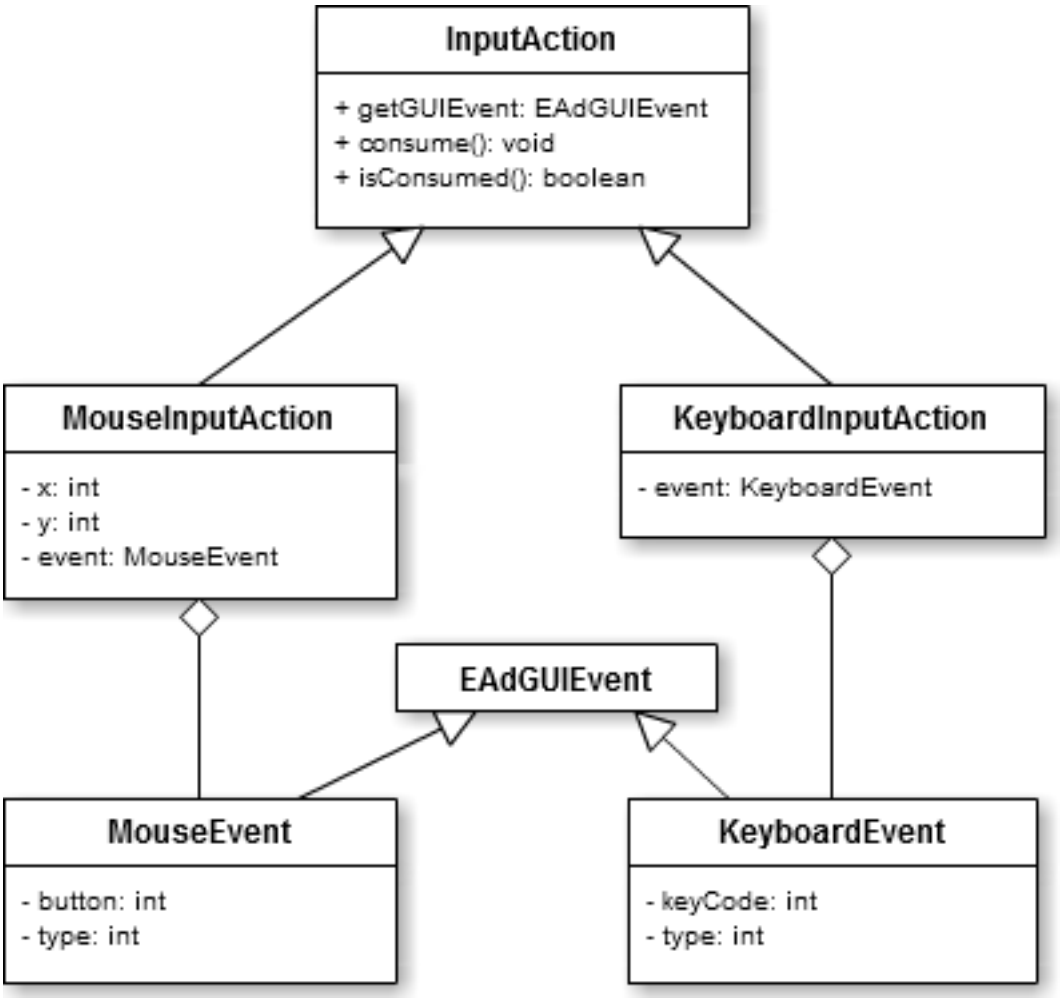

Figure 7.3: *UML diagram for the InputAction class and its inherited classes. All input actions contain the information about the event that triggered them.*

values with the relevant information about every effect.

Deciding these values and the transformation to *LogicTrace* is done on the game engine side, i.e., unlike input actions, which are approximately the same for every game engine, types and arguments in *LogicTrace* totally depend on the game engine. This could be a problem when it comes to generate reports, since these reports are game engine dependant.

However, some standard types of *LogicTrace* are defined, in order to allow some standard reports (see 10):

- **start game**: this type of trace is generated every time the game starts. It has

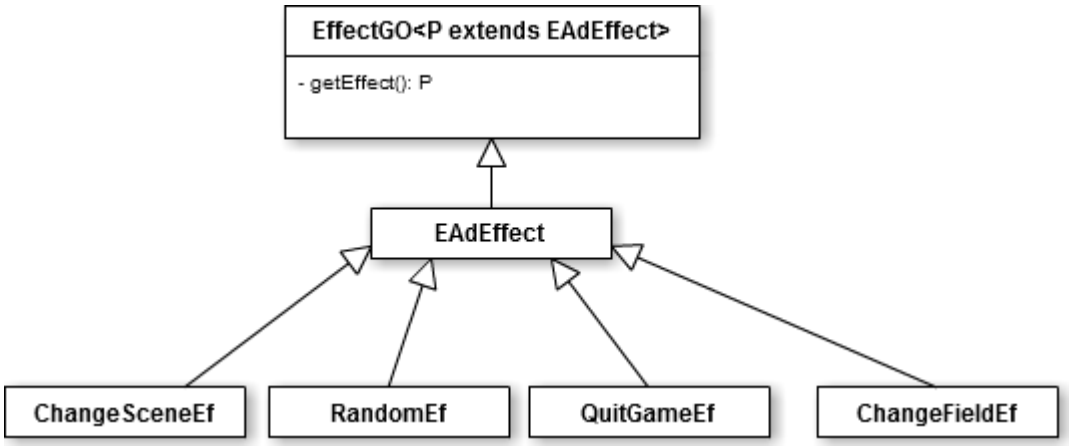

Figure 7.4: *UML diagram for the EffectGO class and its fields. All effects game objects contain the effect data defining them.*

timestamp zero, and as first and only argument receives a timestamp representing the moment in which the game started.

- **end game**: this type of trace is generated every time the player successfully completes de game. If the game can be completed in several ways, the first argument contains and unique id identifying which of those ways was used by the player.

- **quit game**: this type of trace is generate whenever the player quits the game, without completing it. The first argument contains an id identifying where the player was in the game when she abandoned (e.g. a phase id).

- **change phase**: this type of trace is generated when the player changes from one phase to another. The completed phase appears in the first argument, and the next phase identifier appears in the second argument.

- **change var value**: this type of trace is generated when some significant variable changes its value. First argument contains the variable's name and the second argument contains the new value it was assigned to. Concatenation of such traces result in the third proposed type of trace, those containing information about the GameState.

These traces are shown and organize with predefined GLAS reports. However, eAdventure generates more specific traces types which needs non-standard reports in order to show the obtained information to the user. This will happen for every game engine generating non-standards traces.

Chapter 8 details how these effects are captured in the eAdventure game engine.

7.4 Initial filtering

In the first tests it was clear that the initial traces selection done by the GLAS Selector, presented in section 7.2, were not adequate and could not be effectively used. It required to convert every InputAction and every EffectGO into ActionTrace and LogicTrace in order to filter incoming traces. This was highly inefficient because most of traces were finally discarded during selection.

Thus, another selector similar to GLAS Selector, but filtering eAdventure InputAction and EffectGO, was added in order to perform a first filtering process. This class is part of the eAdventure game engine and its UML diagram is shown in figure 7.5.

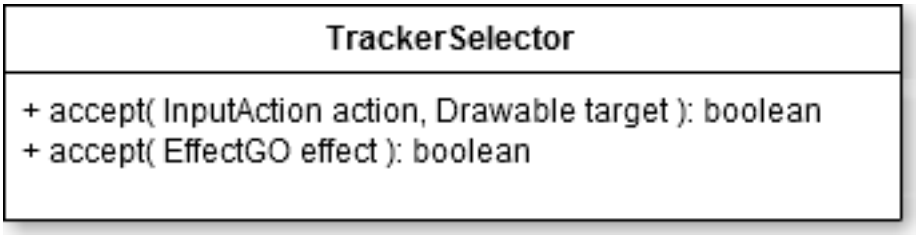

Figure 7.5: *UML diagram for the TrackerSelector inside eAdventure game engine. It has two methods which decide if effects and input actions are selected or not.*

As with GLAS Selector, criteria for selection are defined in two separated configuration files, with similar syntax. Those InputAction and EffectGO that passed the first filter, are converted into GLAS traces and passed to the GLAS Selector.

Chapter 8

Capturing traces and server communication

As presented in chapter 7 about how to do the trace generation, the initial steps in the Learning Analytics process begins in the game engine. Selection and capture steps directly depends on the used game engine. It must be that game engine who actually capture the traces.

However, GLAS framework tries to provide the proper tools and classes to make these steps as simple as possible. In the selection step, a GLAS Selector capable of filtering traces was provided, though in the final implementation another extra selector, integrated in the game engine, was added to make an initial selection based on eAdventure objects.

Classes and tools provided by GLAS for trace capturing are explained in this chapter, and also how these tools are integrated in the eAdventure game engine.

8.1 GLAS Tracker

The GLAS Tracker has three distinct functions:

- **Capture and store**: once they are captured, the traces are stored in a temporary cache before being sent to the server. How traces are captured depends only on the used game engine. Game engine must communicate the generated traces to the GLAS Tracker.

- **Filter**: the filtering process is performed inside the GLAS Tracker. The GLAS Tracker includes a GLAS Selector to run this task over all captured traces. Only those traces accepted by the filter are stored in the temporary cache.

- **Send the traces**: Finally, all stored traces are sent to the server. The GLAS Tracker includes a sender capable of communicating with the server.

Developers should only be aware of how traces are passed from the game engine to the GLAS Tracker, since the other functionality is completely implemented by the GLAS framework.

The class provided for tracking, shown in figure 8.2, has two methods. The first one tracks LogicTrace and the second one tracks ActionTrace. Developers must ensure that their game engine calls these two methods when needed. These calls must use as parameters traces also generated by the game engine. The eAdventure integration case will be presented in section 8.2.

Filtering mechanism was already explained in section 7.2. Sending traces and server communication is presented below:

8.1.1 Server communication

Any communication process between two machines must be managed by some protocol. This protocol must define the steps and the rules for the communication. This section details the communication protocol followed by the GLAS Tracker and the GLAS Server, as well as the implementation of that communication in three different platforms.

Communication protocol

Server side implementation will be detailed in chapter 9. For the following explanation, the internal processing in the server is ignored. It is assumed that the server answers with the correct data to all of the GLAS Tracker requests.

Communication protocol steps are represented in figure 8.1. These steps are:

1. **Start track**: Player starts the game. If tracking is enabled, GLAS Tracker is initialized with the proper parameters. GLAS Tracker sends to the server the user's credentials and an unique id identifying the game the user is playing.

   (a) If the server does not recognise the game identifier or the user's credentials are invalid, it does not authorized the GLAS tracker to send traces. GLAS Server sends a message to inform about what happened. GLAS Tracker disables the trace tracking and the game is normally executed.

   (b) If the server recognises the game identifier and the user's credentials are valid, it sends an authorization message to the GLAS Tracker. GLAS Tracker starts the tracking. Besides the authorization itself, this authorization message also contains required information by the GLAS Tracker, such as the user session counter for that game, or the REST API URIs, required for sending traces. These URIs will be cover with more detail in chapter 9.

2. **Sending traces**: Once the GLAS Tracker has been authorized, it begins to collect and send traces. During this phase, the GLAS Tracker stores all traces generated by the game engine. Once its cache is completely filled, compacts all traces in a single message and it sends it to the server. GLAS Server responses indicating if everything was OK or if there was some kind of error. GLAS Tracker behaves in two ways, depending on the received error:

   (a) If it is an recoverable server error, GLAS Tracker temporary stores the message and waits some time before sending it again.

   (b) If it is an unrecoverable server error, GLAS Tracker aborts the communication. Traces stop being collected and the game continues its execution normally.

3. **End of communication**: When the game ends or if the player quits the game before completing it a final trace is sent. This trace contains a timestamp and marks the

end of the communication between the GLAS Tracker and the GLAS Server. Again, this trace must be generated by the game engine. GLAS Tracker merely identifies this trace, stops the tracking and sends the remaining traces stored in the cache to the server.

Communicating with the Server: implementation

Due to the multi-platform nature of the eAdventure game engine, GLAS offers two possible implementations for the GLAS Tracker. One, implementation is used in Android devices and in Java implementations. The other is implemented with GWT and is used in the HTML5 implementation.

Actually, all these tools, modules and implementations only pretend to ease developers work integrating the GLAS framework with any game engine. But, for other platforms not considered, it is sufficient to implement the server communication through the REST API presented in the chapter 9.

Classes hierarchy for the GLAS Tracker is presented in the figure 8.2.

Both implementations extends the same class, *AbstractGLASTracker*, which implements the base interface GLASTracker. GLASTracker defines all the basics functionalities required to carry out the communication between tracker and server. AbstractGLASTracker is a basic implementation containing all the common features and functions for all the platforms implemented, such as all the operations related to cache storage.

As shown in the UML diagram, GLAS Tracker needs to receive as parameter the server url, a game key identifying the game to be tracked, and a listener that handles the server response. Sending the user's credentials is not mandatory since anonymous tracking is allowed. In this case, users are identified by their I.P.

Implementation changes in the direct communication with the server, which is totally platform-dependant. The solutions for every platform are presented below:

Java and Android Two separates libraries, one for every platform, are used for the server communication. The client part of Jersey library is used in Java implementation. On the other hand, Android uses the HttpClient API, capable of sending GET and POST request as well as receiving responses.

GWT (HTML5) For the HTML5 and JavaScript client, additional restrictions must be considered, in order to provide a tracker with enough flexibility.

The *same origin policy* [13] must be taken into account. An origin is defined by the scheme, host and port of an URL. When the client and the server are hosted in the same origin, they are allowed to directly communicate through AJAX. This mechanism is wrapped in the GWT class *Request*.

However, if client and server are not hosted in the same origin, they are not allowed to communicate through AJAX, due to the same origin policy. To solve this problem, several solutions have been proposed. One of them is to configure the server to accept requests from another origins; other solution is to use a cross domain technique using the JavaScript function *postMessage* and one hidden *iframe*. The details of this hack is explained with more detail in [2].

Whatever the case, GLAS offers the two implementations: one using the GWT request class (with traditional AJAX communication) and another using the *postMessage* technique.

8.2 Tracker integration in the eAdventure game engine

As previously mentioned, traces tracking depends directly on the game engine implementation. It must be the game engine who generates and send the traces to the GLAS Tracker during game play in order to start all Learning Analytics process.

In this section, integration between the GLAS tracking system and the eAdventure game engine is explained for the two types of traces: ActionTrace and LogicTrace.

8.2.1 Catching ActionTrace

In section 7.3.1 user input processing was briefly explained for the eAdventure game engine.

A more detailed scheme is shown in figure 8.3

Native events (Java events, Android events, JavaScript events) are translated into eAdventure input actions, and these are passed to the InputHandler. InputHandler does two different things with these traces: it sends them to the game loop, where this input action fires game effects, and it sends them to the initial filter, integrated in the game engine (see chapter 7).

Once the first filter is passed, input actions are converted into GLAS traces and are sent to the GLAS Tracker, where it all the process already described begins.

8.2.2 Catching LogicTraces

Transformation from effects game objects to logic traces was briefly introduced in section 7.3.2. A more detailed vision of the process is shown by the figure 8.4.

Input events, besides producing Logic Traces, also fire effects. These effects are the minimum unit producing changes in the eAdventure game state, i.e., they represent those events that move forward the game. These effects must be interpreted as LogicTrace.

Effects are intercepted in the GameState and passed to the initial filter. If the trace passes the filter, it is transformed into a Logic Trace (section 7.3.2) and it is passed to the GLAS Tracker.

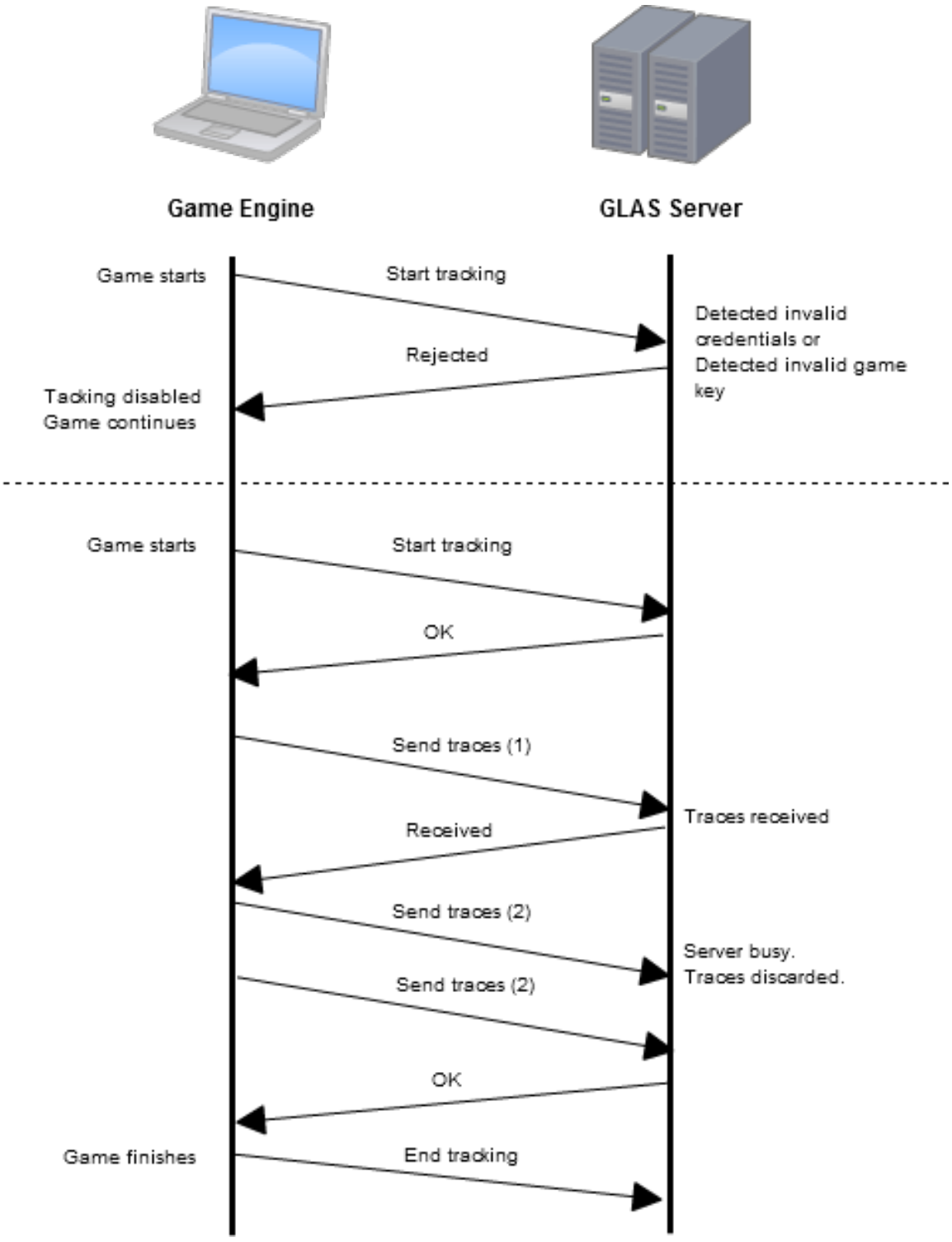

Figure 8.1: *GLAS Tracker communication protocol between the game engine and the GLAS Server*

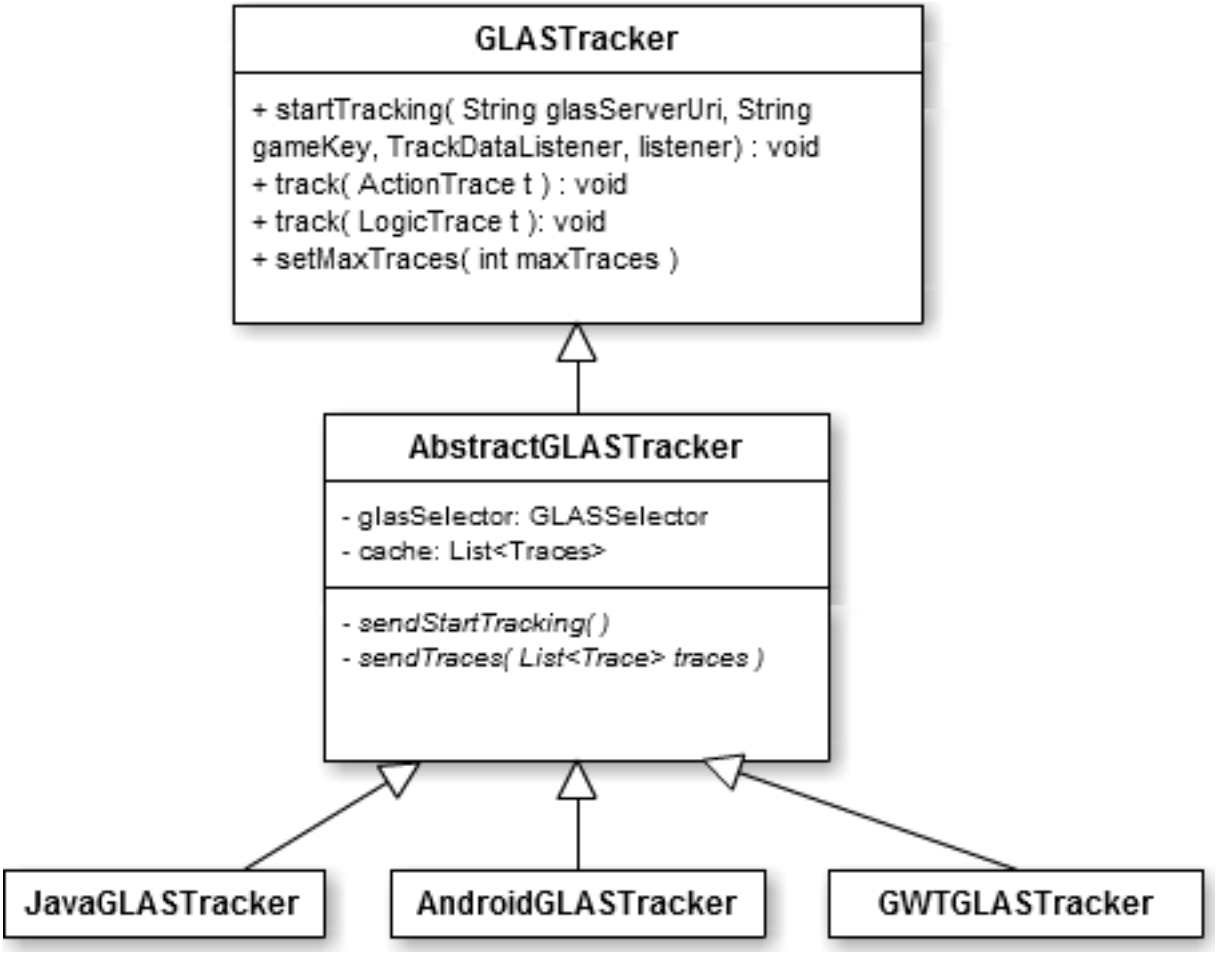

Figure 8.2: *GLAS Tracker hierarchy*

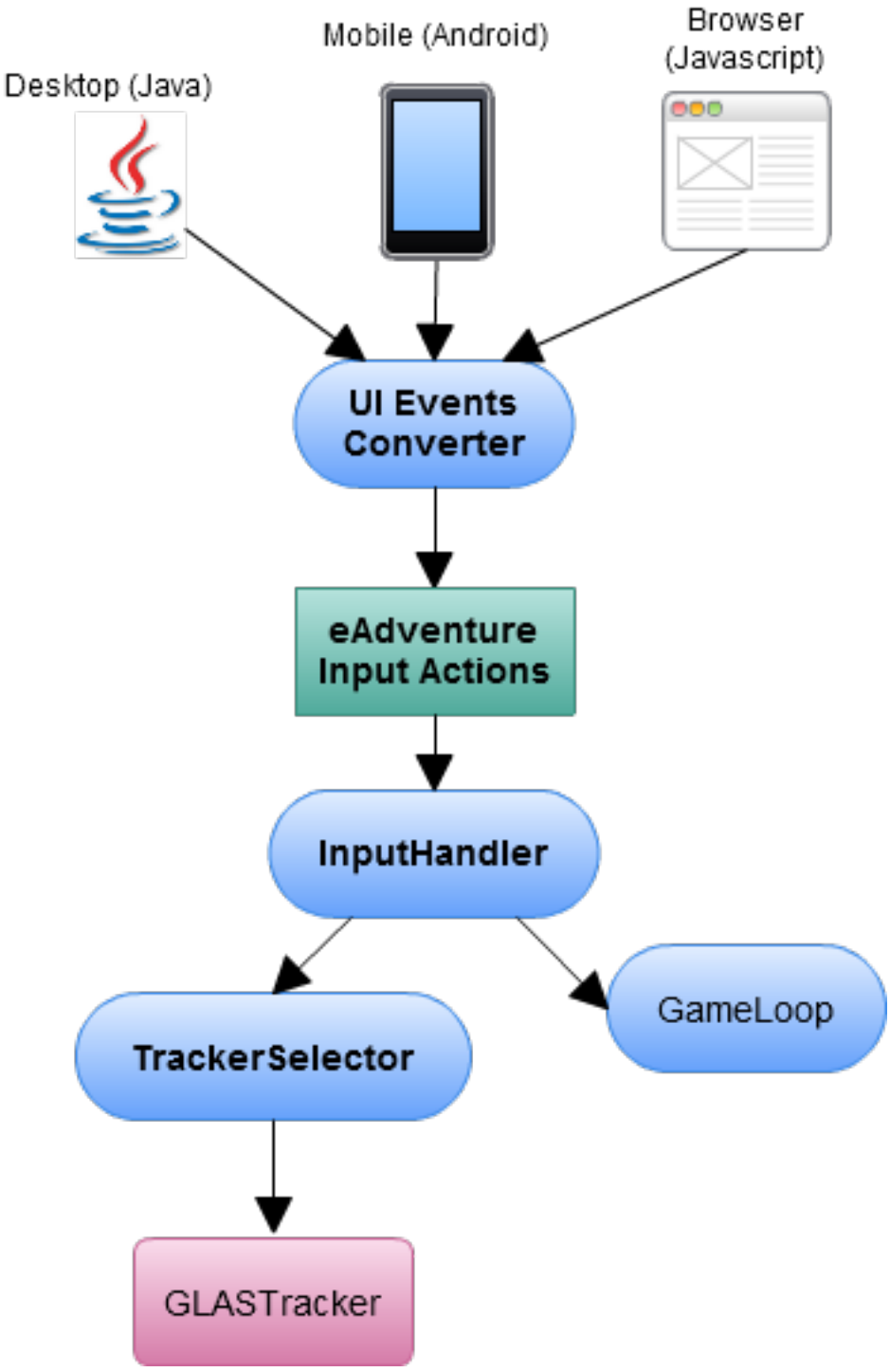

Figure 8.3: *Catching Action Traces in the eAdventure game engine*

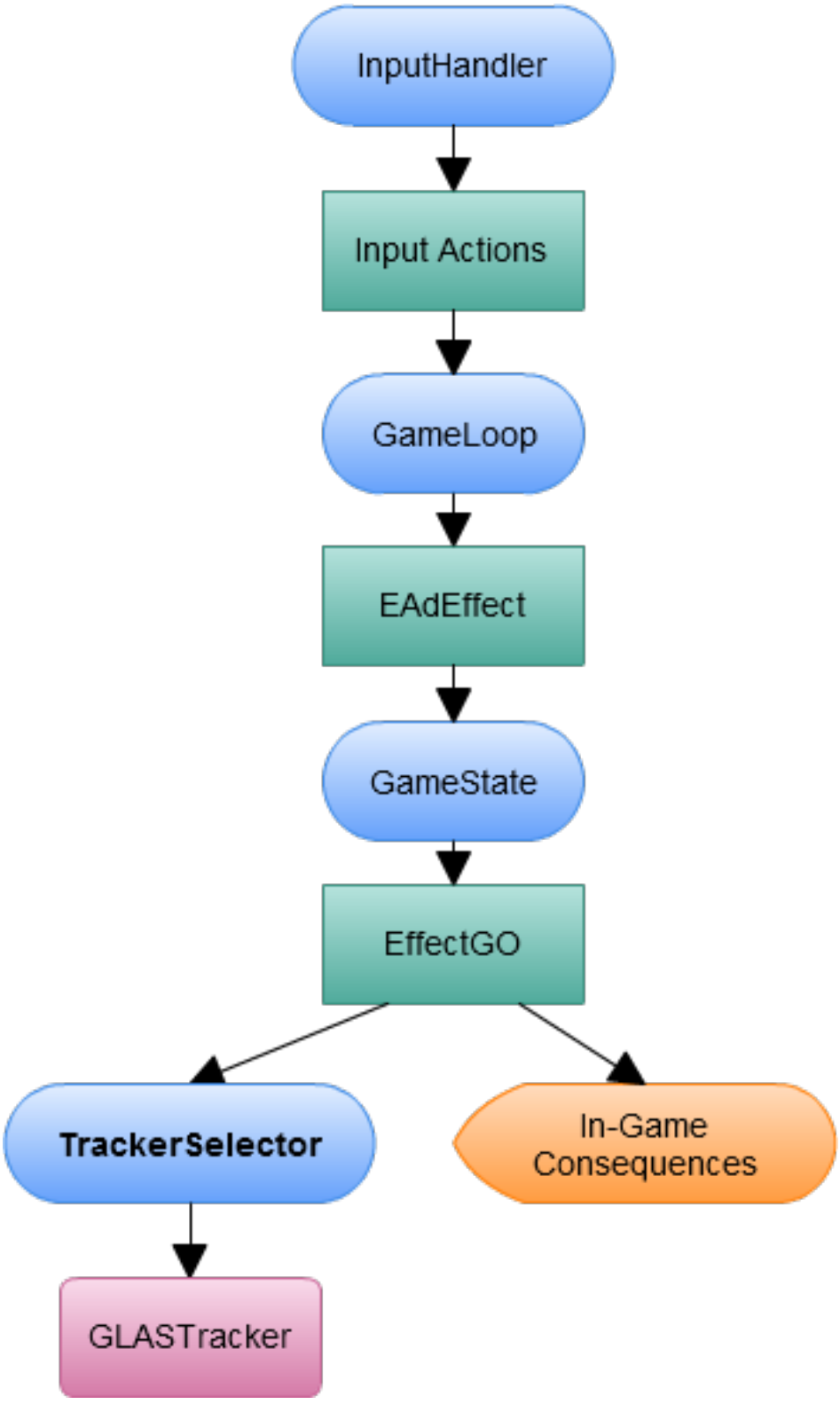

Figure 8.4: *Catching Logic Traces in the eAdventure game engine*

Chapter 9

REST API and Server Implementation

This chapter details the REST API defined to access and store traces in the GLAS Server and its implementation using the Java library Jersey. Database structure and deployment is also addressed, as well as all the server technology involved in interacting with it.

9.1 REST API definition

REpresentational State Transfer (REST) is a style of software architecture for distributed systems such as the World Wide Web. REST has emerged over the past few years as a predominant Web service design model. REST has increasingly displaced other design models such as SOAP and WSDL due to its simpler style [11].

A REST API offers a set of resources identified by URIs. These resources usually represent pieces of data. Four operations can be performed on them:

- **GET**: to obtain the data contained by the resource.
- **POST**: to add new pieces of data.
- **DELETE**: to remove data.
- **PUT**: to update data already stored.

The GLAS REST API defines several types of resources. Some of them allow GET operations only but others allow for GET and PUT operations. DELETE and PUT operations

have not been implemented for any resource because GLAS resources' deletions and updates are never performed through the REST API.

Access to GLAS REST API is made through URIs beginning with:

```
url-server/r
```

Thus, all resources URIs have the previous URI as a prefix. The following resources are defined by the GLAS REST API: Games, traces, game users and queries.

9.1.1 Games

A game is considered as the entire set of traces collected for some individual game, including all the users who played it. It also contains key data about the game itself. The resource identified by the URI

```
url-server/r/games
```

represents a list with all the games being tracked by the GLAS Server.

GET operation returns the game list. The list contains a set of games, each of them with the following fields: an unique identifier (used internally by the database), the game title, the game key (used by the GLAS Tracker to request tracking authorization) and a boolean field indicating whether the tracking is enabled for the game.

POST operation over this URI allows to add new games to be tracked by the GLAS Server. The only data required for creating a new game is its title. The GLAS Server takes care of creating the game key and sets the default boolean tracking value to *true*. Once is created, the game is added to the game list. Game data can be accessed with the GET operation.

Data format used in these operations will be detailed in section 9.2.

Each game is as well an individual resource. Its URI is constructed according to the following scheme:

`url-server/r/games/{gameId}`

The parameter *gameId* is the unique identifier given to the game by the database. The only operation allowed for this resource is the GET operation, which returns the individual game with the fields already mentioned.

9.1.2 Traces

The URI identifying each game is also the root to access the traces collected for them. The URI for traces resource follows the next scheme:

`url-server/r/games/{gameId}/traces/{traceType}`

As said before, *gameId* is the unique identifier - set by the database - of the game whose traces want to be accessed. The *traceType* parameter can take two values: *action* that identifies a resource containing a list of all ActionTrace collected for the game; and *logic* that identifies a resources containing a list of all LogicTrace.

Traces resources allow GET and POST operations. GET operations are used by the GLAS Reporter to retrieve games traces data. POST operations are performed by the GLAS Tracker, in order to add new game traces. Normally, these POST operations contains more than one trace.

For example, consider the following resource's URI:

`url-server/r/games/2/traces/action`

if GET operation is performed over it, all ActionTrace collected by the game with identifier 2 would be returned. The result would be a list of traces. Each of these traces is defined by the fields presented in section 7.1.

If a POST operation, containing as data some captured traces, is performed over the URI by the GLAS Tracker, the GLAS Server receives the sent traces and stores them in the database, linked to the game with identifier 2.

Similarly, consider the following URI resource:

`url-server/r/games/2/traces/logic`

GET operation returns a list with all LogicTrace tracked for the game 2. Each of these traces contains all the fields defined in previous sections for LogicTrace. If the REST API receives a POST operation with collected LogicTrace as data, the GLAS Server stores them in the database, associated to the game 2.

9.1.3 Game users

Additionally, each game contains all users that played them. Game users resource is represented by an URI with the next structure:

`url-server/r/games/{gameId}/users`

This URI represents a list with all of the users that played the game identified by the *gameId* parameter. The list contains game users represented by three attributes: a unique user identifier, the game id and a user session counter (counts how many times the user played the game). The user identifier can be used to obtain more data that are not exposed by the REST API.

This resource only allows GET operations to obtain the users information. Game users are automatically generated by the GLAS Server from "start traces", so POST operations are not implemented for this resource.

9.1.4 Queries

Most of the times, report generators are not interested in accessing the raw data returned by most of the resources. They usually need a subset of that raw data, filtered and prepared for the report intended to be created.

One option is to obtain all data at once and perform the filtering and preparation afterwards, according to the reports needs. However, the GLAS REST API allows a simple query language which enables a more organized access to the data.

These language is based on SQL features and its basic syntax presents the next scheme:

`url-server/r/{resource}/q?c=columns&w=conditions&g=groupby&o=order`

A special "query resource" is accessed adding the extra suffix */q* to any of the previous described resources. After this, query parameters are added as normal GET parameters. These parameters are:

- **c**: Defines the columns that should be included in the result. In this context, columns actually refers to the resources' fields. Syntax for columns is a list of words, one for every desired field, separated by commas. If any of the specified fields does not exist, an error is thrown and nothing is returned. As in standard SQL, asterisk (*) can be used to retrieve all the columns. Additionally, commons grouping functions in SQL, as count, sum, avg, max or min can be used as columns values.

- **w**: it defines conditions to be met by the traces in order to be included in the query result. These conditions are pairs attribute-value, separated by commas, which are concatenated in a single AND condition. For example, if conditions has as value *type,'scn',arg1,'start'* only those traces with value "scn" in field "type" and value "start" in field "start".

- **g**: list columns names, separated by commas, that are used as the "group by" clause in standard SQL.

- **o**: a list with pairs column-order, defining the order in which query results will be returned. For example, if the value were "type,asc,arg1,desc", query result will be ordered first by the column "type" in ascending order and then ordered by the column "arg1" in descending order.

All previous resources returns objects or lists of objects from classes like games or traces. This query special resource always returns a query result object, represented in figure 9.1.

As shown in the figure, returned object contains an array of QueryColumn class. This QueryColumn class contains the column name, a boolean defining if the data is text (if this

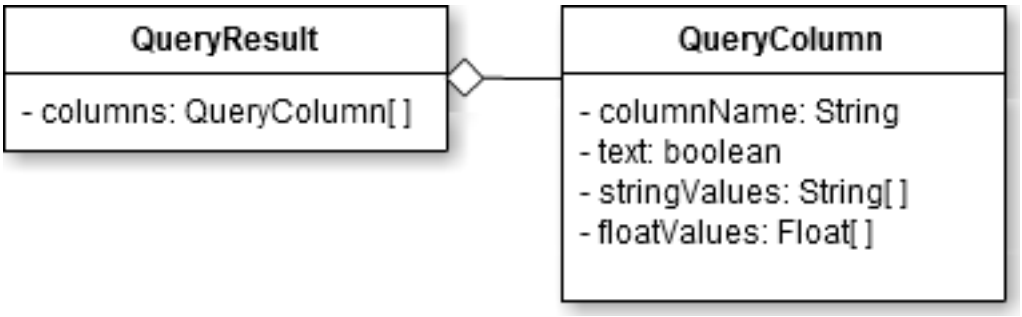

Figure 9.1: *QueryResult contains a QueryColumn for every column consulted in the query.*

boolean is false then data are numbers) and two arrays containing the values for the column Only one array will be filled with data, depending on the column type.

This kind of query can be done over any type of resource already defined by the GLAS REST API, and the result is always a QueryResult object (which can be represented in several formats described in section 9.2).

GET operation returns a response with a QueryResult object and POST operation is not implemented since it has no meaning for this type of resource.

9.2 Data format

Until now, most of objects used in communications are represented by Java classes (as trace definitions in chapter 7). However, due to the undefined nature of the platforms that could interact with the REST API, format for these data must be as standard as possible to ease the communication.

GLAS framework offers three data formats to communicate with the GLAS Server:

9.2.1 XML

GLAS Server is able to send and receive Java objects serialize into XML format, using the Java Architecture for XML Binding (JAXB) [4].

In this architecture all fields of the object are converted into XML tags. The following snippet shows an example of some Java object converted into XML.

```
 <glossary><title>example glossary</title>
  <GlossDiv><title>S</title>
   <GlossList>
    <GlossEntry ID="SGML" SortAs="SGML">
     <GlossTerm>Standard Generalized Markup Language</GlossTerm>
     <Acronym>SGML</Acronym>
     <Abbrev>ISO 8879:1986</Abbrev>
     <GlossDef>
      <para>A meta-markup language, used to create markup
languages such as DocBook.</para>
      <GlossSeeAlso OtherTerm="GML">
      <GlossSeeAlso OtherTerm="XML">
     </GlossDef>
     <GlossSee OtherTerm="markup">
    </GlossEntry>
   </GlossList>
  </GlossDiv>
 </glossary>
```

As will be shown in section 9.3, GLAS Server is able to convert Java objects into XML and send them to clients, as well as converting back XML data into Java Objects, using the Jersey library.

9.2.2 JSON

JavaScript Object Notation is a lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate. It is also based on a subset of the JavaScript Programming Language[7].

With a similar mechanism to generate XML, Java Objects are serialized into JSON. The following data would represent a Java object in the JSON format:

```
{
    "glossary": {
        "title": "example glossary",
"GlossDiv": {
            "title": "S",
"GlossList": {
                "GlossEntry": {
                    "ID": "SGML",
"SortAs": "SGML",
"GlossTerm": "Standard Generalized Markup Language",
"Acronym": "SGML",
"Abbrev": "ISO 8879:1986",
"GlossDef": {
                        "para": "A meta-markup language.",
```

```
"GlossSeeAlso": ["GML", "XML"]
                    },
"GlossSee": "markup"
                }
            }
        }
    }
}
```

Again, in the server side, the Jersey library deals with conversion between Java objects and JSON.

9.2.3 JSONP

JSONP is a variation over JSON that enables access to the REST API data avoiding the same origin security policy in browsers.

Essentially, returned data are the same as in JSON, only that these data are wrapped with a callback function. Something like the following is obtained:

```
callback( {
    "glossary": {
        "title": "example glossary" } );
```

The script HTML tag is not affected by the same origin policy, so it can be used to invoke REST APIs in other domains, and process the data in the "callback" function, whose body must be defined in some other script.

An extra parameter, named callback, must be added to the URI resources to request JSONP data format to the GLAS Server. Callback parameter value must be the name of the function which will process the obtained data.

9.3 Server implementation

To implement the API presented in the previous section, first a database to store all the data collected is required, and second, a server technology that allows to access that information through the REST API.

In this section, database deployment is explained as well as its main tables. Then, it is detailed part of the server technology implementing the REST API.

9.3.1 Database

A MySQL database is used as datastore. Tables scheme is shown by the figure 9.2.

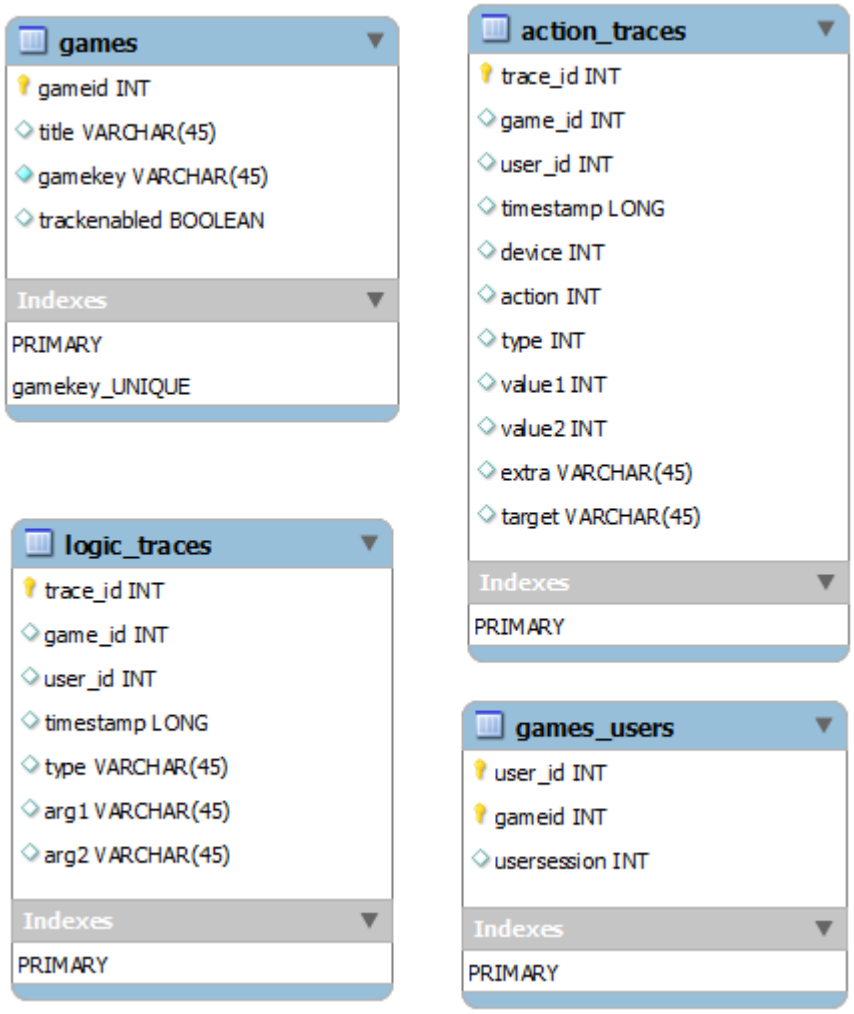

Figure 9.2: *Tables in the GLAS Server Database*

As shown in the figure, data is stored in three different tables:

- **Games Table**: contains all data about games. It has a column for every field previously mentioned in the game resource definition.
- **Logic traces**: table containing all the logic traces collected for all games and users, with a column for every field defined in LogicTrace.

- **Action traces**: table containing all the action traces collected for all games and users, with a column for every field defined in ActionTrace.

Database access

The Data Access Object [1] pattern is used to access the database. A general UML diagram, representing the whole sub-system, is shown in figure 9.3.

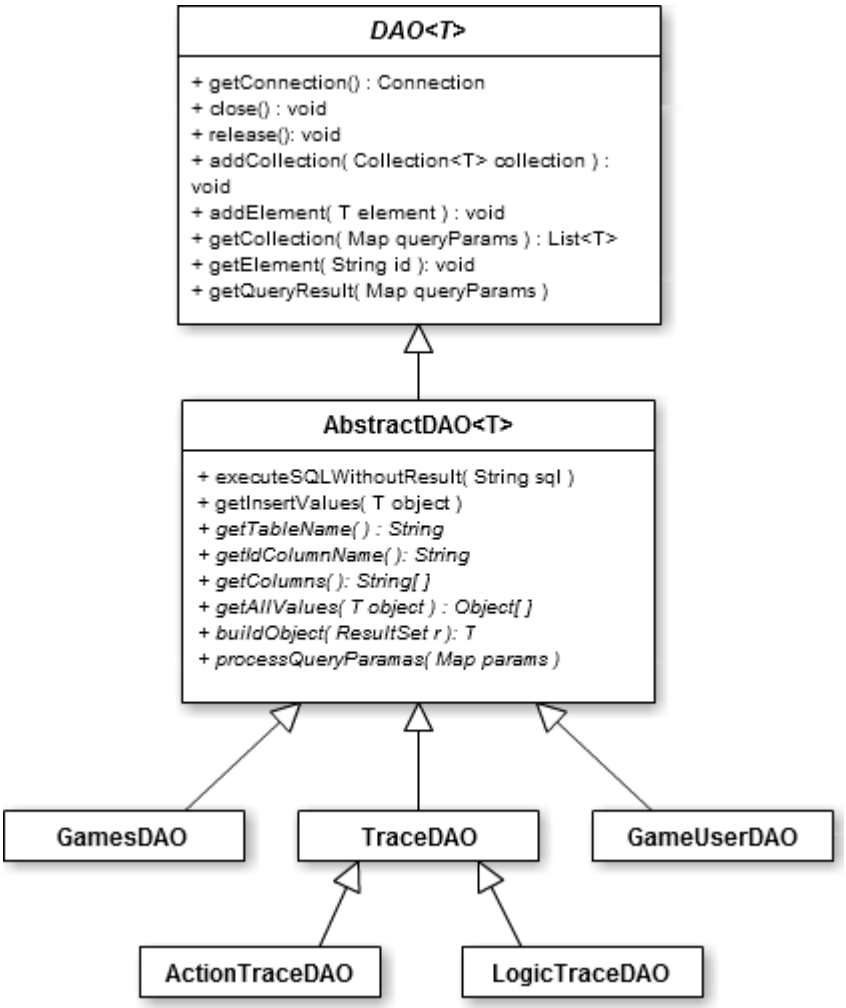

Figure 9.3: *UML diagram showing the Data Access Object pattern at its implementation in the GLAS Server.*

The root interface DAO contains all the function required by the operations allowed in the REST API: methods *addCollection* and *addElement* are designed to support POST

operations sending data; methods *getCollection* and *getElement*, to support GET operations asking for raw data; and method *getQueryResult*, to attend GET operations sending queries.

Additional methods *getConnection, close, release* deal with the direct connection to the database, and release the acquired resources when they are not longer needed.

AbstractDAO extends the *DAO* interface, and it is programmed to deal with SQL databases. It contains several helpers to perform common SQL operations (select, insert, update) and it defines some abstracts methods to be implemented by concrete classes, depending on the type of the resource accessed (marked by the generic type T in the abstract class definition).

Thus, several implementations are provided for each of the possible resource previously presented. These classes only implements the abstract methods required by *AbstractDAO*, which are mostly methods that returns information about the table containing the resource (table name, columns names...). All the processing and work with the database is done by the abstract class.

9.3.2 Accessing to resources

Jersey is used as main library to deploy a REST service in the GLAS Server. Jersey uses class annotations to create resources based on URIs. The steps for creating REST resources with Jersey are:

1. Create a class for the resource and annotate it with *Path* and use as value for the annotation the resource's URI.

2. Create methods to response those REST operations used by the resource, and annotate them with *GET, POST, DELETE* or *PUT*, as required.

3. Tell Jersey where the resources are in the web server configuration file.

For example, the following code:

```
@Path(/games)
public class GameResource {
@GET
@Produces(MediaType.APPLICATION_XML)
Object getXML(){
// ...
return xmlResult;
}

@GET
@Produces(MediaType.APPLICATION_JSON)
Object getJson(){
// ...
return jsonResult;
}

@POST
@Consumes(MediaType.APPLICATION_JSON)
public void put( T object ){
// ...
}
}
```

could represent a handler for the game resource. Path annotation value (concatenated to the server URL) defines its URI, and GET and POST annotations define which methods must be executed when GET and POST requests are received. Additionally, these methods can even define the data format that produces/consumes. Jersey automatically invokes the right method, depending on the information contained by the HTTP request.

The class hierarchy shown in figure 9.4 represents all the resources defined in section 9.1.

GameResource, GameUserResource, ActionTraceResource and LogicTraceResource handle the resources defined in 9.1, and have *Path* annotations with their corresponding URIs.

All of them extends AbstractResource, the class which actually has defined all the GET and POST operations. Inherited classes only define the Path annotation and some of them override the *initResource* method to meet their particular needs.

Additionally to resource access, there are two special types of resources:

Install

Accessing the URI

`url-server/r/install`

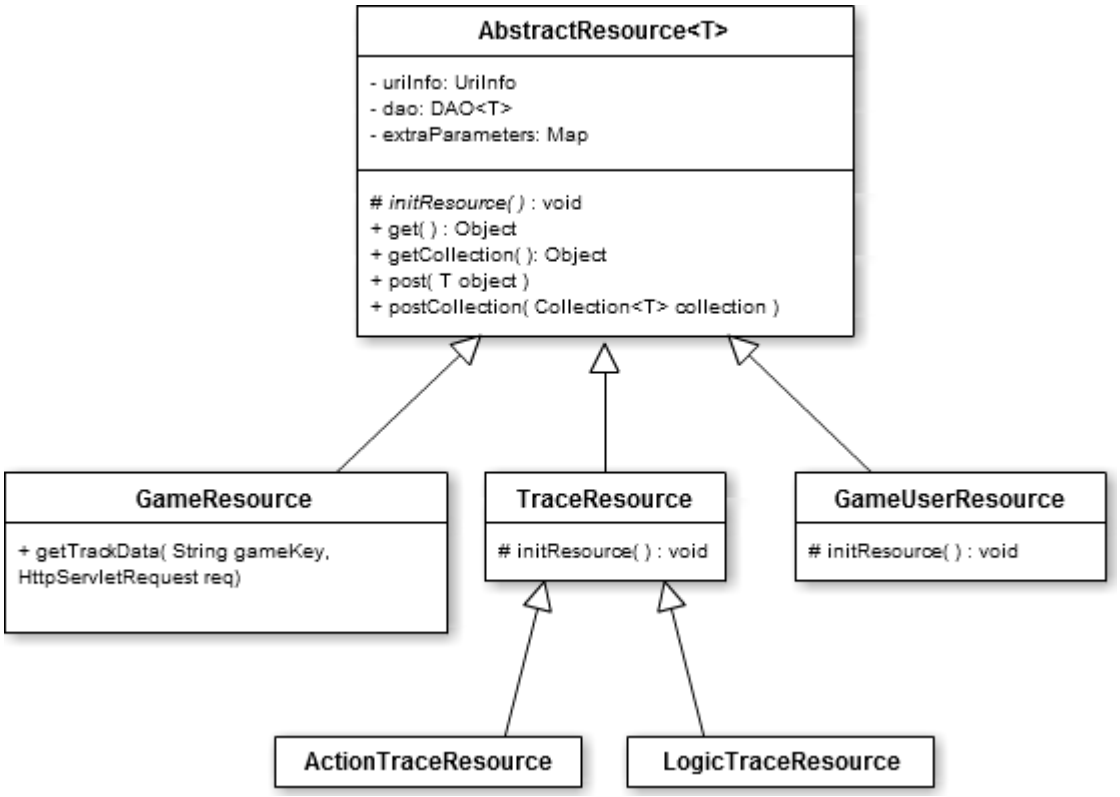

Figure 9.4: *Architecture deployed to provide access to all GLAS resources.*

installation process for the GLAS Server is started. Essentially, this "resource" creates the database structure and the initial data, among other tasks. The only information required is the database location.

Start tracking

As said in section 8.1.1, communication between GLAS Tracker and GLAS Server begins with a message sent from the GLAS Tracker to the GLAS Server in order to obtain an authorization to send traces.

This message is a POST request sent to a special resource defined by the following URI

```
url-server/games/track
```

GameResource checks the game key and, if any, the user credentials, and sends back a proper response.

Chapter 10

Generating reports

Report system is designed to work in Internet browsers. GLAS Reporter is programmed with the Google Web Toolkit to allow its deployment in most modern browsers.

These chapter explains the general architecture used to generate reports. Then, what types of reports exist and what kind of data are shown in every of them.

This is a first approach to a more complex reports system. This system will integrate all reports presented in a unique web application. This application will allow to navigate for the different games and check results of students and group of students.

10.1 Reports architecture

In GLAS, a report is defined as a graphic representation of some data. These data proceeds from a query result returned by a GLAS REST API request. Reports can be generated using games collective data, showing total or average results, or using single users data, showing individual results.

General architecture for reports is presented in figure 10.1.

All reports extends the GLASReport interface. This interface defines three methods:

- **load(gameId, gameUser)**: Begins the report generation. It receives as parameters the game id and the game user id, if any, to generate the report.
- **getApiCall(gameId, gameUser)**: it returns the URI of the GLAS resource used

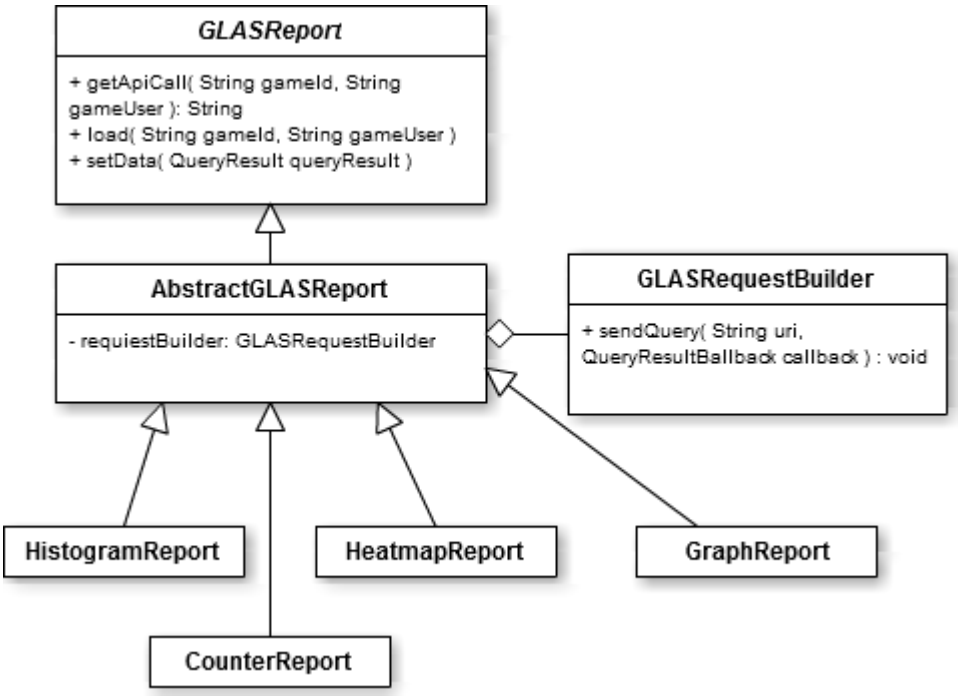

Figure 10.1: *UML diagram showing the reports architecture and some of the reports types*

to generate the report. It receives the game id and the game user as parameters.

- **setData(queryResult)**: this method is called once the request is done and the resulting data are received. The query result parameter contains all the data required to fill the report.

The abstract class AbstractGLASReport deals with the communication with the server, using a GLASRequestBuilder. Inherited classes takes the data received from the request and represent it.

10.2 Types of reports

The current GLAS representation includes the following types of reports:

- **Counters**: Shows a number representing some quantity. It is used, for example, to show the number of users of some game, how many times some user played a game... (Figure 10.2)

- **Histograms**: Shows a barchart with some data. It is used to show times spent for every user in complete the game, compare time spent in every phase by user on average... (Figure 10.4)

- **Heatmaps**: Shows an image with colored areas representing hotspots. The meaning of these hotspots depends on the context. For example, heatmaps are used to show the most clicked areas on a specific game phase.

- **Graphs**: Shows a graph made by nodes and connections. For example, this type of report is used to represent the path followed by an user phase to phase in a game. (Figure 10.3)

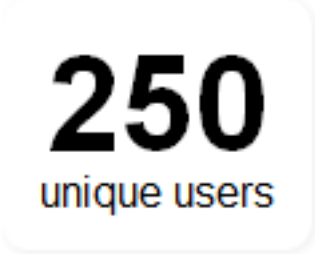

Figure 10.2: *A report with a counter. In this case, it is used to show the unique users that played the game.*

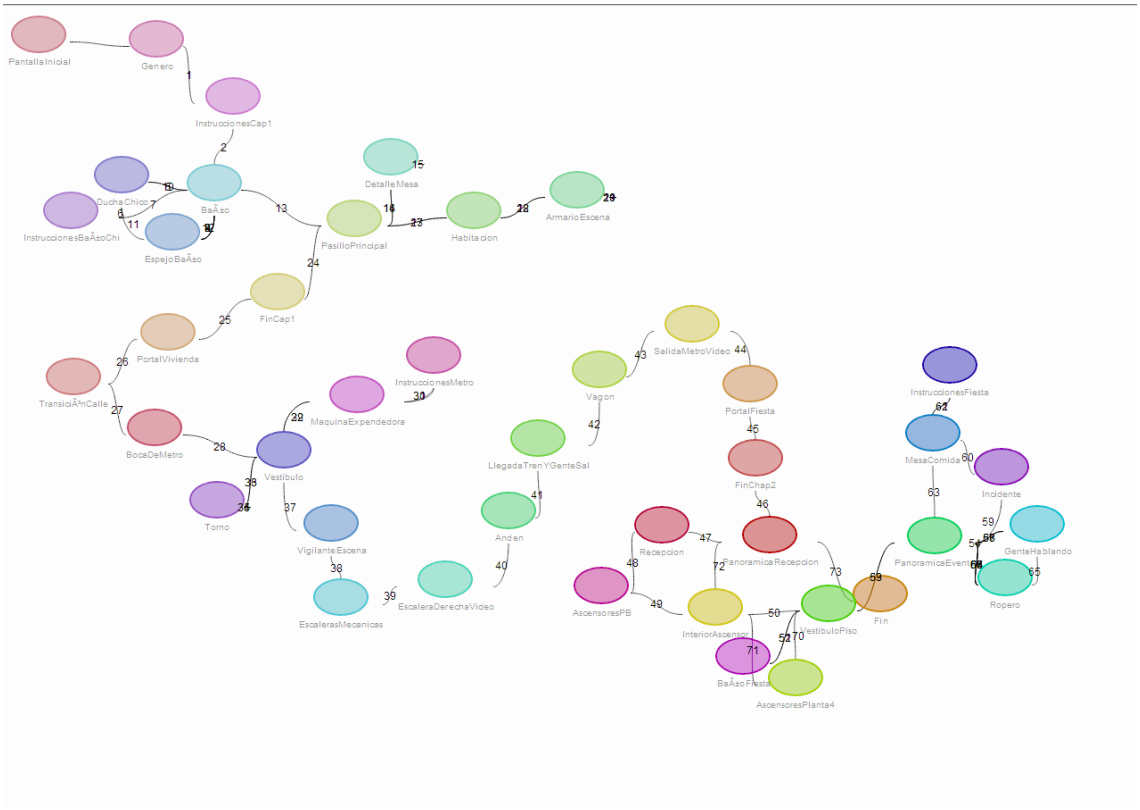

Figure 10.3: *An example a graph report. In this case, it is showing the exploration path followed by a player in a specific game.*

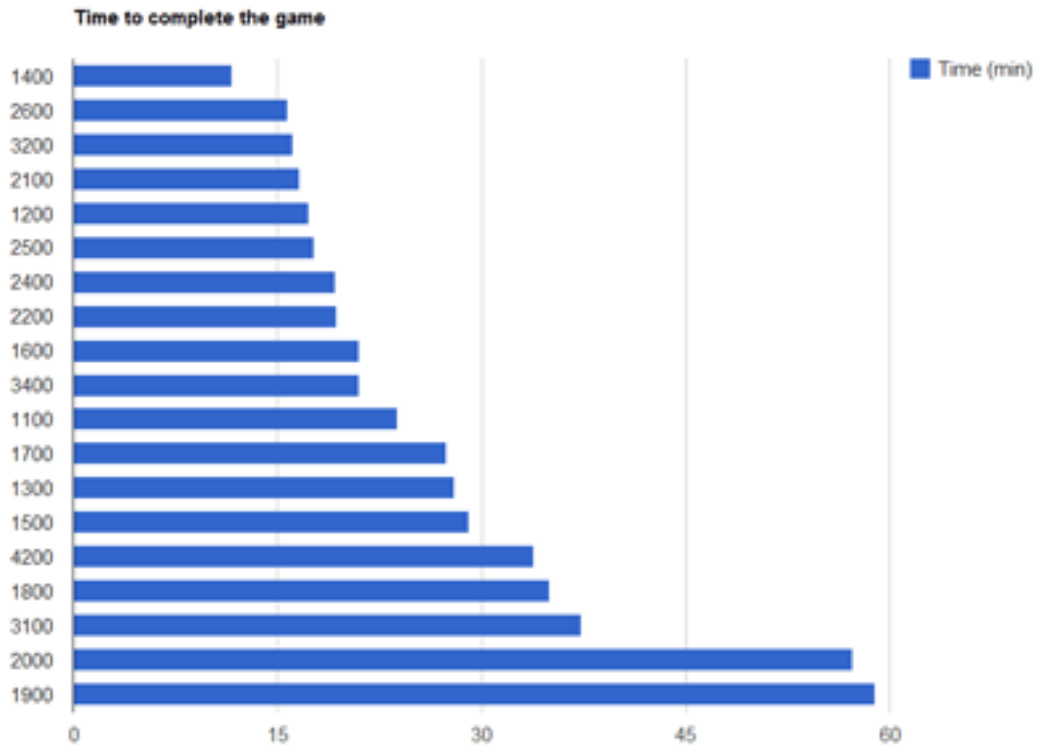

Figure 10.4: *An example of histogram report. In this case, it represents time spent for every user on completing a specific game.*

Part III

Conclusions

Chapter 11

Conclusions and future work

This chapter discusses what are the main contributions of the project and draw some conclusions about the work carried out. Then, some future work planned for the project is also presented.

11.1 Contributions

The main contribution of this work is the proposal of an abstract model to perform the Learning Analytics process over educational videogames. This abstract model covers all Learning Analytics Steps: it defines very generic game traces that could be applied to almost every game genre; it also defines how these traces could be filtered and stored; it proposes several standard visualization methods for stored data and several assessment rules to evaluate students performance.

A specific implementation has been developed to validate the abstract model. This implementation has resulted in the GLAS framework. The framework uses eAdventure as game engine and consists of three separate subsystems: the GLAS tracker, responsible for selecting and capturing all traces generated by the game engine; the GLAS Server, responsible for some trace aggregation tasks and for storing all collected traces; and the GLAS Reporter, responsible for some additional aggregations tasks and for showing data visualization to the users.

The abstract model proposes two additional subsystems: an evaluator to assess the

students according to their game play, and a game *adapter* capable of reacting in real time to players game play. This *adapter* intervenes in the game flow in order to adapt the game to the player needs. These subsystem were designed but the implementation was not completed due to project's time constraints.

Finally, the main contribution of this work have been reflected in two conference papers:

- *A framework to improve evaluation in educational games*, Serrano-Laguna, A., Marchiori, E. J., del Blanco, A., Torrente, J., and Fernandez-Manjon, B. (2012). Proceedings of the 2012 IEEE Global Engineering Education Conference (EDUCON) (pp. 1-8). IEEE. doi:10.1109/EDUCON.2012.6201154. The complete publication is included in appendix A.

- *Tracing a little for big Improvements: Application of Learning Analytics and Videogames for Student Assessment*, Serrano-Laguna, A., Torrente, J., Moreno-Ger, P., and Fernandez-Manjon, B. Presented to the 4th International Conference on Games and Virtual Words for Serious Applications (VS-GAMES'12). Submitted and pending on final decision. The complete publication is included in appendix B.

Even if the complete GLAS framework was too large to be completely implemented in the given time, the framework is ready to track games, store data and generate reports. Besides developing the framework itself, another extra elements were needed.

First, specially designed educational games are required in order to fully test the framework. Some pre-existing eAdventure games were used to carry out some initial tests, but these games were not designed to contain a complete assessment process. The extracted data were only useful and enough to generate simple reports. Serious assessment to be effective should be present in the early design of games.

Users trials with the games were also required. When the track system was ready, it was too late to start tracking some games with actual users in order to obtain substantial data, and perform some deeper statistical analysis.

Now, all efforts can be focused on create games and perform in-depth analysis that will be used to improve the implementation and to fully validate the abstract model.

11.2 Future work

GLAS framework just begin with the first steps if applying Learning Analytics to games, even when GLAS collects the data, stores it and generate reports about it.

However, the actual power of Learning Analytics is much higher. Reports are good for visualization, but the actual goal is to find models able to predict results, to assess students in a rigorous and reliable way and to provide concrete actions designed to improve the whole educational process at any level.

Future work, from a scientific viewpoint, includes:

- **Better data analysis**: analysis performed over the data are too simple to extract all the complexities hidden in data. More complex statistical analysis could be used in order to extract more relevant information and to discover relationship between the different metrics.

- **Test if data collected is enough and adequate**: traces collected for videogames mainly derived from game mechanics and from intuition. Better data analysis could confirm or refute if the selected data is adequate for the intended analysis. Even if the type of data collected was right, it still remain the problem of the initial selection, which input events to track, which logic events to log... Some general rules for data selection could be also extracted analysing more videogames.

- **Information visualization**: information visualization is a key process for understanding large amounts of data. Current reports elaborated by GLAS are pretty simple, since the data obtained is also pretty simple. When more advanced analysis methods are performed, more complex data could be extracted and new ways of visualizing that information will be needed.

On the other hand and from a more technical perspective, GLAS framework needs to evolve including some of the following features:

- Implementing some secure protocol (e.g. like OAuth [9]), to restrict the access to the GLAS REST API and to identify users in a safe and secure way.
- Improve framework security. Since it is a prototype version, different security risks usually encountered in web applications have not been fully addressed.
- Improve data storage performance. In the current implementation, all messages containing data to be stored that are received by the GLAS Server create a database connection. Using a share queue to store the incoming data and a process taking the elements in the queue and adding them to the database would allow much more simultaneous connections, which is fundamental to improve the performance of this type of systems.
- Create a integrated report web application that allows to navigate for all the games and reports generated by GLAS, using users with different roles (teachers, students, administrators...).

Bibliography

[1] Core j2ee patterns - data access object. http://java.sun.com/blueprints/corej2eepatterns/Patterns/DataAccessObject.html.

[2] Cross domain post with postmessage. http://benalman.com/projects/jquery-postmessage-plugin/.

[3] Databases market share. http://www.mysql.com/why-mysql/marketshare/.

[4] Java api for xml binding. http://jaxb.java.net/.

[5] Jersey: Restful web framework for java. http://jersey.java.net/.

[6] Jfreechart: Java chart library. http://www.jfree.org/jfreechart/.

[7] Json: Javascript object notation. http://www.json.org/.

[8] Loco-analyst: Learning analytics software. http://jelenajovanovic.net/LOCO-Analyst/index.html.

[9] Oauth: an open protocol to allow secure api authorization. http://oauth.net/.

[10] Reload: Reusable elearning object authoring and delivery. http://www.reload.ac.uk/editor.html.

[11] Restful web services: The basics. http://www.ibm.com/developerworks/webservices/library/ws-restful/.

[12] Restlet: Restful web framework for java. http://www.restlet.org/.

[13] Same origin policy in web browsers. http://www.w3.org/Security/wiki/Same_Origin_Policy.

[14] Server-side technologies survey 2010. http://www.webdirections.org/sotw10/server/.

[15] Snapp: Social networks adapting pedagogical practice. http://research.uow.edu.au/learningnetworks/seeing/snapp/index.html.

[16] Tonic: A restful web app development php library. http://peej.github.com/tonic/.

[17] E. J. Marchiori I. Martínez-Ortiz P. Moreno-Ger B. Fernández-Manjón Á. del Blanco, J. Torrente. Easing assessment of game-based learning with <e-adventure> and lams, 2010.

[18] Richard Blunt. Does Game-Based Learning Work? Results from Three Recent Studies, 2007.

[19] J Chen. Flow in games. *Communications of the ACM*, 50(4):31–34, 2007.

[20] L. Communiqué. Towards the european higher education area: Responding to challenges in a globalised world, 2007.

[21] T. Elias. Learning analytics : definitions , processes and potential., 2003.

[22] Rebecca Ferguson. The State of Learning Analytics in 2012 : A Review and Future Challenges a review and future challenges. *Media*, (March), 2012.

[23] M. Hassler. Web analytics., 2010.

[24] Eugenio J. Marchiori Pablo Moreno-Ger Baltasar Fernández-Manjón Javier Torrente, Ángel del Blanco. <e-adventure>: Introducing educational games in the learning process, 2010.

[25] A. Dix K. Gilleade. Using frustration in the design of adaptive videogames. *Proceedings of the 2004 ACM SIGCHI International Conference on Advances in computer entertainment technology ACE 04*, 74:228–232, 2004.

[26] H Willis A Levine K Haywood L Johnson, R Smith. The 2011 horizon report, 2011.

[27] M. F. Paulsen. Experiences with learning management systems in 113 european institutions., 2003.

[28] S. Williams and N. Williams. The business value of business intelligence, 2003.

Part IV

Appendices

Appendix A

A framework to improve evaluation in educational games

Ángel Serrano, Eugenio J. Marchiori, Ángel del Blanco, Javier Torrente, Baltasar Fernández-Manjón

Department of Artificial Intelligence and Software Engineering

Complutense University

Madrid, Spain

aserrano@e-ucm.es, [emarchiori, angel.dba, jtorrente, balta]@fdi.ucm.es

Abstract — The evaluation process is key for educator's acceptance of any educational action. The evaluation is challenging in most cases but especially when educational games are used. In educational games if in-game evaluation exist it is usually based on a series of simple goals and whether these goals are achieved (i.e. assessment). But we consider that evaluation can be improved by taking advantage of in-game interaction, such as the user behavior during the game and the type and number of interactions performed by the user while playing. In this paper, we propose an evaluation framework for educational games based on in-game interaction data. We discuss how user interaction data is collected in the most automatic and seamless way possible, how to analyze the data to extract relevant information, and how to present this information

in a usable way to educators so they achieve the maximum benefit from the experience. The evaluation framework is implemented as part of the eAdventure educational platform, where it can be used both to improve upon traditional basic assessment methods (i.e. goals, scores and reports) and to provide information to help improve interaction with games (e.g. discovery strategies).

Learning Analytics; Educational video games; framework proposal; case study;

A.1 Introduction

In traditional education, either in higher education or in other levels, the main evaluation method is based on written final exams [1]. This method, as some authors have pointed out [2], presents a series of problems. These problems are related not only to the student evaluation, but also to the evaluation of the educational action itself: the amount of data available is limited, and it is usually restricted to students and educators subjective perceptions (e.g. through polls about the past courses). Other metrics, mostly based on exam grades, might not give enough information about the educational action, or whether it was a success or a failure and why. Moreover, these data usually become available when the action is finished or when it is too late to make an intervention, improvement or correction in the ongoing action.

With the emergence of the Web, on-line educational resources have grown exponentially. Many institutions now use LMS (Learning Management System) to organize their courses, to allow students to communicate among themselves and with teachers, and to improve access to educational resources [3]. Still, despite all of these on-line resources, evaluation is still usually performed using traditional methods. Most of the content presented in LMS is finally evaluated through written exams in classrooms, or through online tests or exams.

However, there is a whole new body of data, derived from the student interaction with on-line educational resources. These data can be collected and analyzed not only to improve the evaluation methods, but also to obtain real-time feedback about the progress of any

educational action, enabling educators to predict results and react to that progress.

The field studying the use and analysis of this kind of data is known as Learning Analytics [4]. This new field advocates of capturing all the data derived from interaction with on-line educational resources, and analyzing it to assess students, predict future events and act consequently to refine educational actions. These ideas have been successfully applied in other disciplines, like Business Intelligence, a well-extended set of techniques for analyzing business data to support better business decision-making [5], or Web Analytics, where internet data are collected in order to understand and optimize web usage [6].

Currently, LMS are the main target for Learning Analytics systems. Projects like SNAPP [7] or LOCO-Analyst [8] offer statistics about the interactions made by students inside an LMS. SNAPP is focused on analyzing forum activity and creating network diagrams of all interactions among students. From this, it infers which are the most active students in a class, and those students who are "disconnected" or "at risk", among other features. LOCO-Analyst is based on student interaction with learning content (e.g. number of views, time spent with every resource) that is used to infer conclusions about learning content characteristics (e.g. difficulty or importance).

Though LMS activity reports can contribute to better understand students' interactions with content and resources, educational games represent an ideal environment to capture more detailed and diversified student interaction. In the last few years, Game Analytics are being used to let developers know about how players interact with their games. One of their main purposes is identifying where and why a player got stuck during the game , so game developers can try to smooth this hardness, to avoid player frustration and thus keep him engaged and playing [9]. All these ideas applied to educational games, combined with other Business Intelligence and Web Analytics techniques and guided by the Learning Analytics process, are addressed in this paper.

First, we board the main steps in the Learning Analytics process, and how these steps can be particularized in educational games, proposing a theoretical Learning Analytic Model

and a Learning Analytic System. Then, we propose an implementation upon the educational game platform eAdventure and a use case to deploy the system, and finally some ideas and thoughts about the whole process.

A.2 Learning Analytics Steps in Educational Games

Authors agree that Learning Analytics process [2] begins with selecting the most relevant student data to be captured. Once it is captured, data must be aggregated and transformed into reportable information (for example, using charts or other visual representations). With this information, the educator should be able to judge how the student used the educational resource. Some authors call this step predict, since information is converted into knowledge, and knowledge enables predictions. However, in our approach we will use this step to assess the student, and for now on, we will name this step as assess. Assessment information can be used, under certain conditions, to dynamically assist the student, and to refine the educational resource, based on the students results. Finally, all the knowledge acquired can be shared with others whom could benefit from it. Table A.1 represents a scheme with all the steps and their descriptions.

To support all these steps, we propose a system based on a Learning Analytics Model (LAM) holding all the information required for every step, and a Learning Analytics System (LAS) endued with all the processing power required by the model.

In this approach, focused on educational games, we consider the LAS as a separate system from the game engine, but both are communicated. The LAS also has access to the game model and the LAM (Fig. A.1). The LAM is constructed by a set of models, which are directly related to the different modules contained by the LAS (Fig. A.2).

In this section the learning analytics steps are described in general but also for educational games in particular, building the LAM and the LAS upon them. As starting point for this definition we consider the available data in games and how to adapt these data to the theoretical model in order to extract the maximum information possible from this media.

Step	Unit produced	Description
Select	Data	Choose the basis data to be captured
Capture	Data	Collect selected data
Aggregate and Report	Information	Sort out captured data and convert it in information
Assess	Knowledge	Understand reported information and convert it into knowledge, assess students
Use	Knowledge	Adapt the system based on assessment
Refine	Knowledge	Improve educational action
Share	Knowledge	Show knowledge for the benefit of others

Table A.1: *Learning Analytics Steps*

A.2.1 Select and capture

First, the data to be captured by the LAS are selected. These data will be the raw material that will feed the steps that follow. The data selection criteria are lead by the educational resource objectives and some constrictions such as technical limitations and privacy policies must be taken into account. In educational resources, meaningful data can be selected from personal information about the student (e.g. age, gender, etc.), academic information and any other data provided by the resource context.

While in static resources (e.g. PDF files), the only extractable data are the number of views and the time spent with them, the interactive nature of educational games provides a whole new type of data that can be selected:

- GUI events performed by the student during the game: mouse clicks, keys pressed, and other events (joystick movements, buttons interactions), depending on the input method. Not only the event itself can be recorded, but also the time when it occurred and whether it was performed over a target (e.g. some click over a game object). These events can provide clues about the student behavior during the game (e.g. if

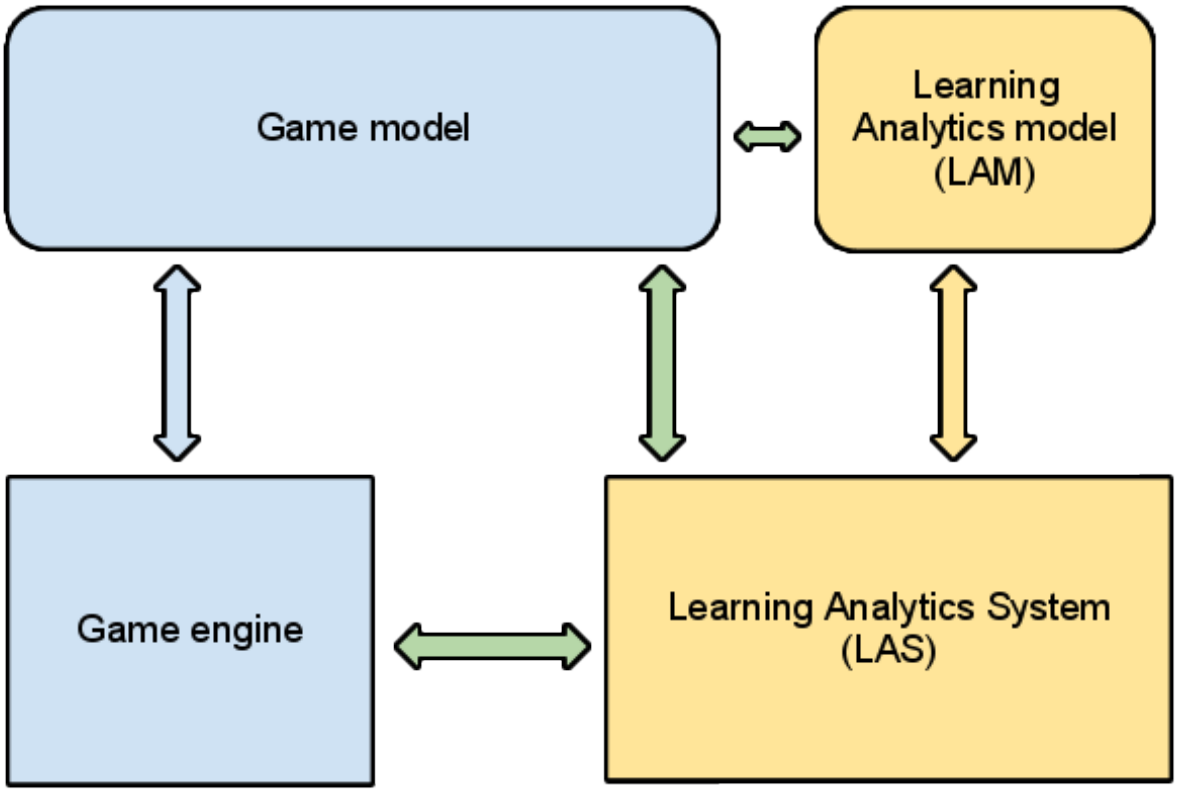

Figure A.1: *Relation between the different components involved in the Learning Analytics process. The LAM is dependent on the game model and the LAS. LAS is aware of the LAM and the game model, and can communicate with the game engine.*

all GUI events were captured the LAS would be able to recreate the complete game play).

- Game state evolution: the game state is a set of variables and their values that specify a concrete status in the game instance. The evolution of variables through time describes the development of the different goals of the game. Depending on the case, the whole game state evolution could be recorded, or it could be recorded only in some points (e.g. when a phase ends, or a goal is achieved).

- Logic events: a logic event is anything that moves the game-flow forward. Changing the value of a variable, finishing a phase, launching a cut-scene (i.e. a slide-show or video), losing a life, achieving a defined goal, etc. Some logic events, and their

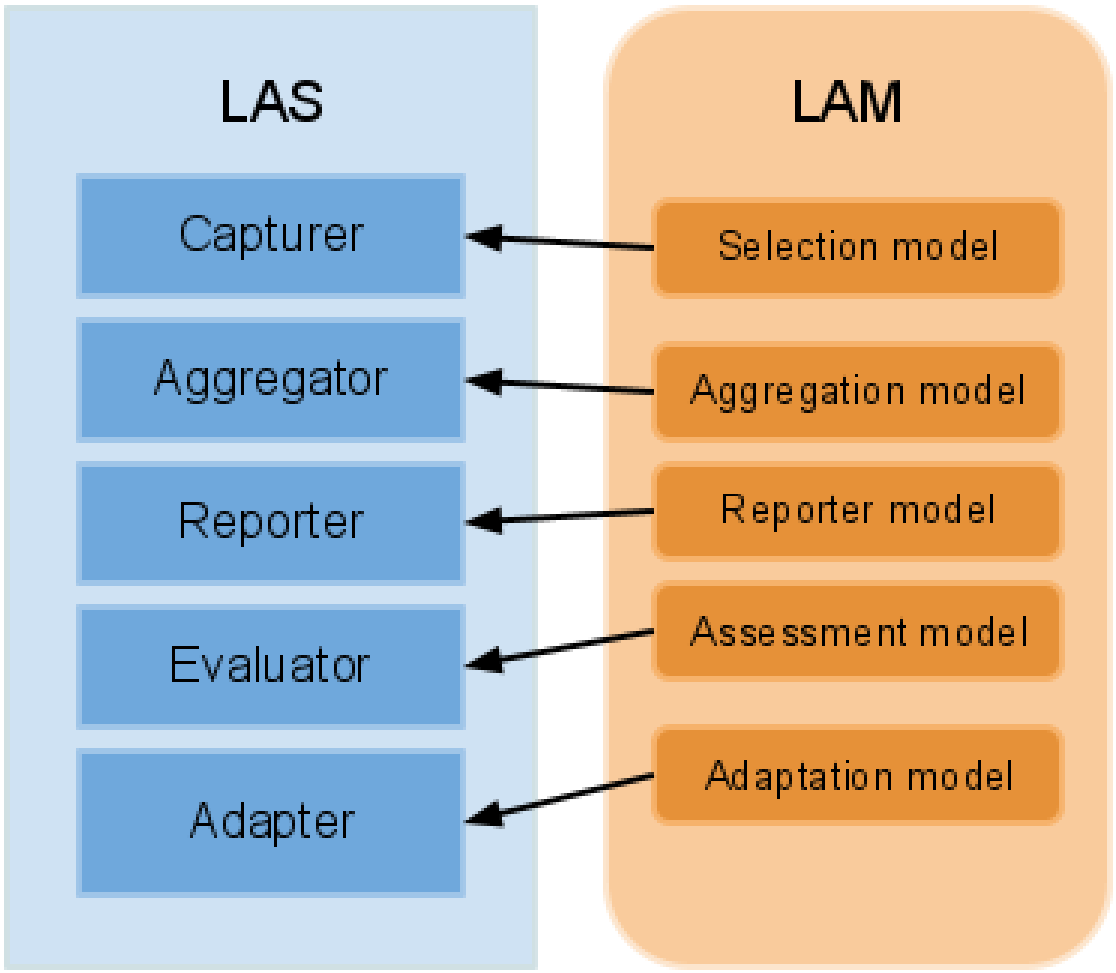

Figure A.2: *The LAS consists of a series modules that take part in the different steps of the Learning Analytics process. The LAM holds models for those steps which requires defining a model to work. The LAS uses the LAM to process all the data captured and generated.*

timestamps, can be directly related to the student progress in the game, and thus be relevant for the assessment.

Selectable data are limited by the technologies used to deploy the games: Not every piece of data here proposed will be available in every game platform. These selectable data, then, are platform-dependent and must be defined in the LAM's selection model. To avoid unnecessary data capture, every game model should define, among all selectable data, the

final data to be captured according to their own purposes.

Once the data are selected, the framework requires a way to capture it. The technology involved will be very important to establish how the data are collected. Access to different internal parts of the game engine is required to capture some of the information. This implies, for instance, that such model cannot be generally applied to commercial games provided as black boxes.

Another issue is the moment when the captured data are passed to the LAS to begin processing. The simplest way is to store all the data locally and send it back to the LAS when the game is finished. Data could be sent in certain significant moments, like when the student ends a phase or achieves a goal. Moreover, all the data could be sent to the LAS as they are being captured. Last two approaches enable real-time assessment that can be used to assist the student during the game. Depending on the needs, all these data might go through a filter in order to make it anonymous.

A.2.2 Aggregate and Report

The captured data must be organized in such a manner that it can be shown in human readable formats, like tables or graphics. A more meaningful report can be done if the LAM contains, in the aggregation model, semantic rules to interpret all the received data. For example, the system could relate a raw event (e.g. a variable taking a particular value) with a meaningful feat (e.g. the player completed a goal). These semantic rules can be based on the game engine, where some events can have an implicit meaning (e.g. an engine where pressing escape key always brings up the menu) or on the game itself (e.g. if the game variable "hits" is "8", the phase is completed).

Semantic rules can be expressed like conditions producing new data to be reported: when a condition (based on GUI events, a logic event or a concrete game state) is met a new unit of data, defined in the aggregation model, is generated. E.g. when in the game state, the variable score equals to 10, and the variable gold equals to 15, the LAS aggregator produces

a logic event "Goal 1 completed". The reporter could then treat this event as it would with any other logic event.

The LAS' aggregator needs to be endued with mechanisms capable of understating and processing these kinds of rules (Fig. A.3).

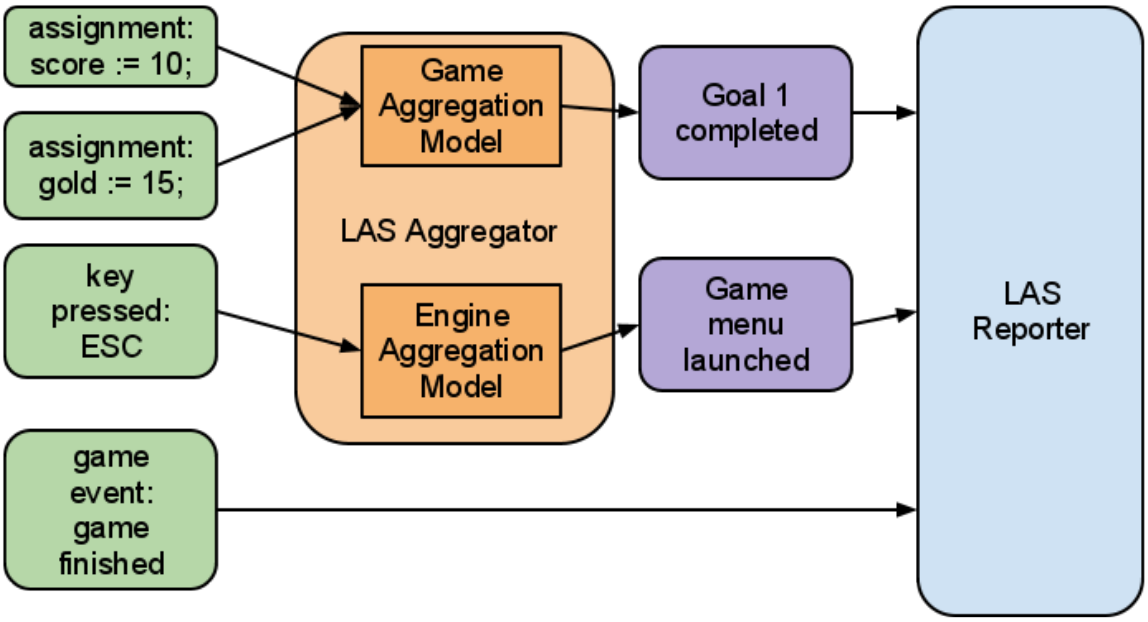

Figure A.3: *Raw events are passed through the semantic rules contained by the LAM, and converted to more meaningful data. Events can be grouped and simplified through semantic rules. Some events, such as the game finishing, have enough meaning and do not need to be interpreted by any rule*

After aggregation, information can be reported in common ways, such as tables or charts, but also, we can take advantage from the inherent characteristics of games to report information with new representations. For example, "heat maps" could be created for every phase, in which the heat can measure the amount of times the player clicked in every point of the phase, or the places where the player was defeated. If there is enough information of user interaction, an animation recreating how the student played a game phase could be shown.

The reporter model contains which information must be reported and which representa-

tions must be used. Common reports can be defined at engine level (e.g. heat maps for every phase can be common for all games), as well as reports at game level, holding important information in that particular game.

These reports can be even richer if data from different students are aggregated. Average results can spot which goals took more time or the places where most of the students failed.

A.2.3 Assess and Use

The information and the reports generated until now can give an overview of how the students are using an educational resource. However, this information should have some practical consequences to be really useful. It is the moment to transform the information received into knowledge. In the educational game context, all the information reported is processed to assess the student in this step.

Games are organized around goals. In educational games, these goals should be based on the success in some educational aspects. In our context, and based on the concepts of the selection and aggregation process, we could have several types of goals, represented by:

- A GUI event or a series of GUI events performed by the student, over a game object or in total.
- A concrete game state, fulfilled fully or partially.
- A variable taking a defined value.
- The launch of a particular logic event.

These classes of goals are platform-dependant, and should be defined by the LAM's assessment model.

Compound goals can be defined based on these simple goals. An educational game can define all the necessary goals to cover all the educational aspects that are to be learned by players of the game. Based on these goals and with the reported information the game can be used to assess the student.

This assessment can have two applications: one, just to measure the success of the student in the game, and act accordingly (e.g. enabling the student to access to new educational resources), or, if the captured data are being passed to the LAS during game time, dynamic adaptation through real-time assessment (e.g. if the student got stuck in some point, the system will offer him help). Rules for this assistance are contained by the adaptation model, and are processed by the adapter, which is able to communicate with the game engine to perform the adaptation.

Assessment and dynamic adaptation could be more sophisticated. As some authors pointed out [10], propagating information through complex structures, like Bayesian networks, can help to determine what is going on in virtual simulations, and better decide what adaptation profile to choose.

However, our approach pretends to be based on easy principles and stay accessible for as many educators as possible. Complex structures, like Bayesian networks, are normally out of reach for most educators.

A.2.4 Refine and share

With all the accumulated knowledge from previous steps, an educator can know about the global results (assessed in the previous step) of the educational action and can identify which educational goals were not achieved as expected. Thus, educators can refine the educational resources to improve the results or readjust their expectations.

In educational games, those game goals that were not solved as expected can be detected. Aggregated data from several students can ease this job, pointing out, in average, which goals made more trouble to students. From here, the game could be modified or even redesigned to facilitate its accomplishment. This does not directly imply making the game simpler or shorting the educational goals, it could be enough to smooth the learning curve in the game, or adding some extra help in game points especially hard. Maybe, learning analytics conclusions showed that the student did not get stuck, but stop playing the game

after a while, indicating that it was not engaging enough.

Finally, the LAS can share all the knowledge obtained with other systems. These systems cover from LMS to institution administrative systems. Even making the data public can be an option in some cases. In order to be able to share data, some considerations such as privacy policies, what knowledge is shared or which standards are used in the communication, need to be taken into account.

A.3 Implementation Proposal: eAdventure

eAdventure is an educational game platform developed and maintained by the <e-UCM> research group at the Complutense University of Madrid for the last 5 years. This platform includes a game engine and an easy-to-use editor, targeted at educators. eAdventure is currently undergoing the development of the 2.0 version, where new features are being added. Some of these include support for multiple platforms [11] and an easy to use narrative representation of games [12]. Moreover, we propose to implement the framework presented in this paper in this new version of the system.

eAdventure games are composed of scenes, which can represent from a simple scenario in an adventure game where the player's avatar moves to a more complex slide-show, going through an array of mini-games and other content. These scenes are always composed of simpler parts referred to as scene elements, each of which will usually have a graphical representation, a position in the screen, behaviors, etc. The current scene and the status of elements in the screen are defined as the game state. It is the flow from one scene to another, behaviors of the scene elements and effects (changing current scene, showing text, launching videos, assigning values to variables) that make up a game, by continually changing the game state until a final state is reached.

Most of the modules forming the LAS are implemented on a server (Fig. A.4). The whole system is initialized with the Game Model, (containing the Adventure LAM) and the Engine LAM. The LAS has several modules to satisfy the requirements for every step of

the Learning Analytics process. Each module is initialized with the information received by the initializer, and remains ready to process all the incoming input. LAS pipeline begins with the data collected by the capturer from the eAdventure game engine. The data is sent to the server, where first is aggregated, according to the rules defined by the Adventure LAM, and second, stored in a database. This information can be shown to users via web through the reporter, and also be used to assess the student, through the evaluator. If adaptation is enabled, assessment data is used by the adapter to change the game state in the eAdventure engine. Finally, LAS can communicate its data to external systems. A more detailed relation between the modules and the steps is detailed below.

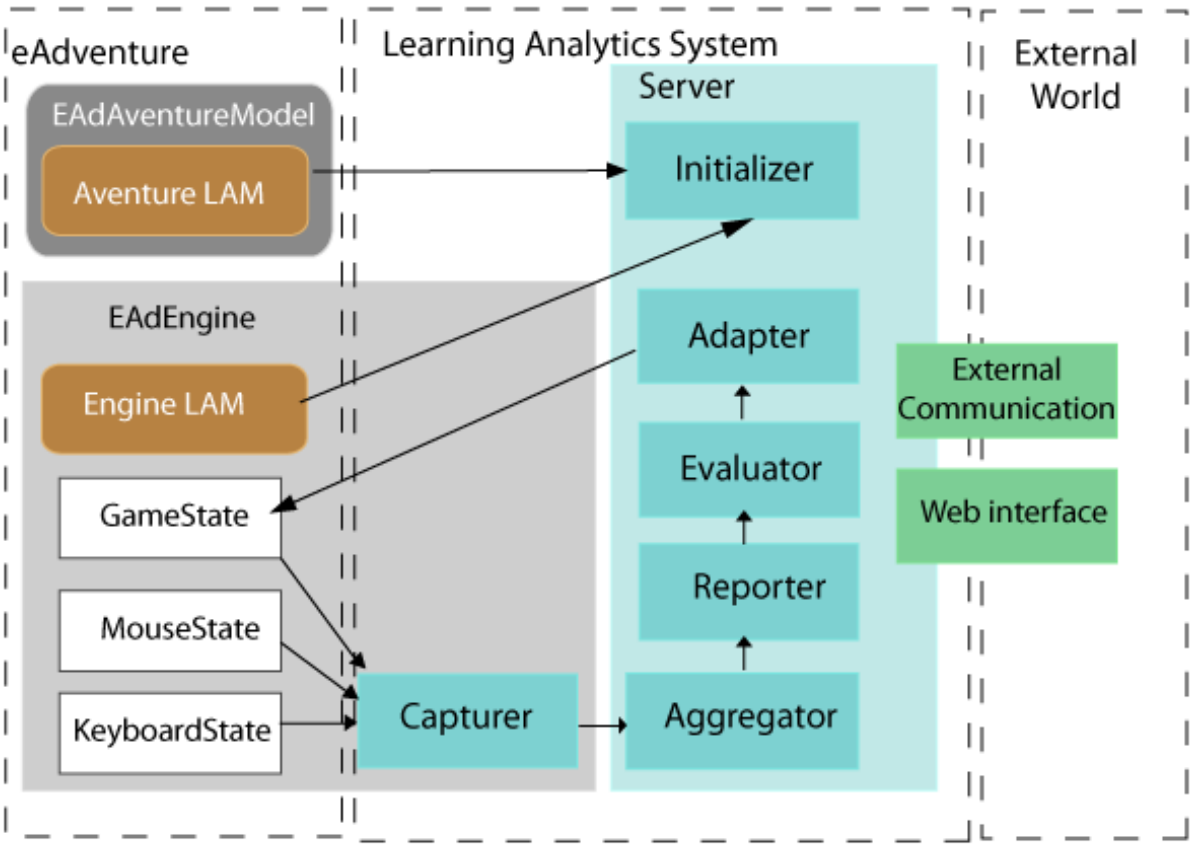

Figure A.4: *General organization for the LAS integrated with eAdventure. The LAS is mostly deployed in a server, and can communicate with the game engine. A web interface and external communication with other systems is offered as well.*

A.3.1 Select

Given that eAdventure is intended to be a general game engine, including its own editor, our proposal tries to make selectable the biggest amount of data, letting to the game designer choose between all the available options. The eAdventure Learning Analytics Model defines three units of selectable data:

- LAGUIEvent: represents a detailed GUI interaction. It holds the GUI event (mouse action, drag and drop, keyboard action) with its properties (mouse button, key pressed) and the target scene element, if exists.

- LALogicEvent: represents the launching of a game effect. It holds the generic effect data and additional information about the concrete instantiation (e.g., in the changing scene effect, the initial scene and the final scene).

- LAGameState: represents a game state in a certain moment. It contains a map holding all the game variables associated with their current value.

Every of these units has associated a timestamp, representing the moment the event occurred since the game was started.

In the editor, we select where and when these data must be captured. For example, we can mark which type of GUI events (mouse, keyboard) we want to capture for every scene element (LAGUIEvent), and there is a special option to record all the GUI interactions performed in the game, allowing the recreation of the whole game play. It is possible also to capture those game effects that have relevance in the game flow (LALogicEvent) and, if desired, produce a LAGameState with the current game state. LAGameStates can be configured to be automatically generated periodically, when a game condition is met or when the game ends.

All these options are added to the selection model as part of the game's LAM, which will be used by the LAS' data capturer.

A.3.2 Capture

To capture all these data it is required a data capturer with access to all the relevant parts of the eAdventure game engine. The game engine has three main elements that are involved in this process: the input listener, which processes all GUI input from the user; the game state, which stores all the values for all the variables in the game at any given point in time; and the effect handler, which processes all in-game events (such as scene changes). All these elements communicate any relevant change to the data capturer, which then captures this information according to the current game selection model. The captured data are instances of the selectable data units presented before.

All these collected data are sent to the LAS aggregator (Fig. A.5). Due to the multi-platform nature of the eAdventure game engine, different implementations take care of the communication with the aggregator. The data capturer can be configured to send out the captured data when the game is finished, a scene change happens, a defined condition is met or in real time.

A.3.3 Aggregate

The data sent are received by the aggregator, who makes a first data processing based on the semantic rules defined by the aggregation model, contained by the Adventure LAM, converting the basic units into more semantic pieces of data. This new units can be defined through the editor, as well as the rules of conversion from basic units (presented in the II.c section).

Aggregator also groups all the GUI events by type and scene element, filters redundancies and stores all the data in the LAS database.

Previous versions of eAdventure provided a basic mechanism for data aggregation and report generation. This basic mechanism allowed for information to be written in a textual

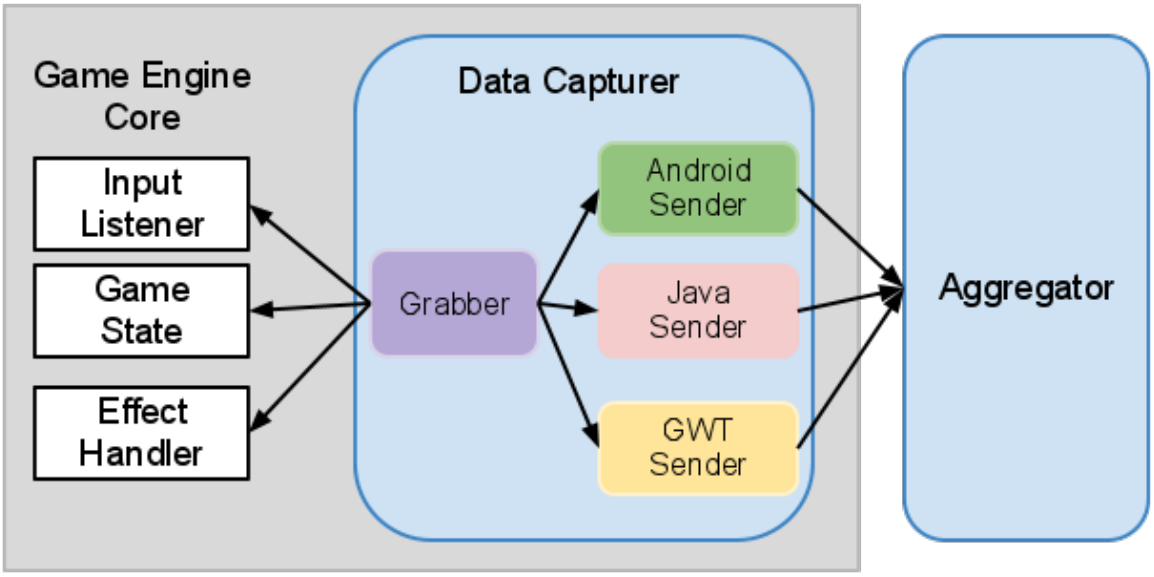

Figure A.5: *The data capturer is compound by two elements: a grabber connected with all the elements producing significant events in the game engine, and a sender managing communication with the aggregator*

report based on the values of variables in the game. This way, the game author could define a set of rules that would, for instance, write in the report that the player failed to complete a goal if a variable to indicate this was given a certain value. [13]

The same sort of data aggregation can be performed with the new LAS system. The rules in this case use a syntax that establishes the meaning of the data that were captured during the game to generate a report. The most basic reports will only include basic textual information, such as "Goal X was completed at time Y". However, more complex information can be aggregated to generate detailed information about the goal, such as "Goal X was completed by time Y, after Z attempts where the player failed to solve problems A, B, C, etc."

A.3.4 Report

The reporter represents in a web format all the information stored in the database. Among many others, the reports can be:

- A table relating scenes with the total time spent in any of them.
- Heat-maps showing where the player is hovering with the mouse most.
- Screen capture recreated from game states.
- Game animations built from captured GUI events.
- Graphics showing the evolution of chosen variables in time.
- Tables with direct queries to the database.

All these data can be shown for every student or for groups of students. The system can be extended to add new reports from the stored data.

The reporter is intended to help teachers and game designers to understand how the student is interacting with the game, showing data that might not be relevant for the assessment, but for improve the game design, or give clues about how to improve the learning experience.

For example, a heat-map report (Fig. A.6) could show the places where students are clicking during the game for every scene. Teachers and game designers could check if the hot spots are the ones they expected and if not, act consequently, for example modifying the game trying to focus students clicks in the intended points.

A.3.5 Assess

As established by the theoretical approach, this step is when the student is assessed. The assessment model contains all the goals established for the game. Goals can be defined in the editor as variables taking certain values at given times. The evaluator takes these goals and checks them against the stored information.

The accomplishment of these goals can be viewed through the reporter, and can be sent to the adapter to enable dynamic adaptation or to be shared with external systems.

Figure A.6: *Heat map showing the concentration of left mouse clicks in a scene. Main heat zones are situated in interactive elements of the scene.*

A.3.6 Use

When the data are being captured in real time, dynamic adaptation can be used in the game. Adaptation rules are defined in the adaptation model. These rules can be defined in the editor and contain:

- An effect, which is considered the adaptation event and could be any eAdventure game effect (showing a text, changing a variable's value, launching a video...)
- A condition, establishing when the effect should be launched. Conditions can be the general conditions offered to create game logic in eAdventure games, or conditions based on goal accomplishment (e.g. a goal is not completed when the time for doing it

expires). The adapter takes the current game state and the goals information offered by the evaluator to check adaptation model conditions. When a condition is met, it communicates to the game engine the effect to be launched.

A.3.7 Refine

To support the refine task the LAS offers, through the web LAS reporter, information about the individual goals. This allows, for instance, the goals that lead to the worst performance to be identified. How the performance of students can be improved based on this is up to the game designer. However, to ease this task, results obtained for the games' goals can be compared through time (checking if results improved after student played several times the game) and between different versions of the game.

A.3.8 Share

Nowadays, the selection and adhesion to standards for the content interoperability is an essential matter in the development of e-Learning contents. Current e-learning standards, like SCORM [14], are not prepared to communicate all the information collected by our LAS with other systems. For this reason, the best way of taking advantage of the full potential of our approach is to develop specific ad-hoc communication solutions for the systems that take into account all these data (e.g. a Moodle plugin). This idea can also be carried out in the eAdventure activity in LAMS [13], where all the information can be gathered and shown to educators, and use them to modify the lesson flow in an automatic or monitored way.

In the near future, it will be feasible to implement our ideas in compliance with next generation standards. For example, one the last initiatives leaded by the IMS Global Consortium, the IMS Learning Tool for Interoperability (IMS LTI), goes in that direction. This specification allows for the execution of learning tools hosted in external servers. Until other promising standards mature [15], our LAS is able to export all the information contained in the database along with the LAM required to interpret it into a exchangeable XML-based

format.

A.4 Use case: Basic math game

We propose using the framework described in this paper in a basic math game targeted at school children. This game covers basic addition, subtraction and multiplication concepts, challenging the students to solve different problems within a given time-frame. This game is intended to provide increasing difficulty in the challenges presented to the students (e.g. the number of digits in the numbers involved is increased gradually).

The main goal of the game is sub-divided into simple goals: learn to operate with numbers of 1, 2 and 3 digits. Each time one of the goals is met, the game is adapted to provide the next level of challenge. Moreover, to provide help to the students and limit the chances that students could get stuck in any particular level, a help button is always accessible. The use of the help button is part of the information selected to be collected by the system.

In the selection model we marked the help button to capture all left-clicks performed over it. We also added to the model all the keyboard interactions, since these will be the input method used by the students to give their answers.

In the aggregation model a rule is added to convert every left-click over the help button into an "ask for help" event. Another rule is added to transform a sequence of numbers typed in the keyboard, followed by an "enter" into an "answer given" event, which its value is the introduced number and if the answer was correct.

The reporter shows the number of times the "ask for help" event occurred and the number of right answers. For the wrong answers, the invalid value introduced by the student is also shown. Average time for every operation is also displayed.

In the assessment model three goals are added: a percentage of all the given answers, after a minimum number of operations, must be correct (without using the help button) for operations with 1, 2 and 3 digits.

In the adaptation model an effect that changes the difficulty level (operations with 1, 2

or 3 digits) is added when the goal for the current event is achieved.

A.5 Final remarks

Learning Analytics, unlike Business Intelligence or Web Analytics, is still an emerging field that has a great potential. In this paper we tried to focus on a single target (i.e. educational games) in order to develop concrete methodologies trying to clarify some of the steps involved and better define the whole process. But, we think most of the ideas proposed for educational games can be extended to analyze data from other types of interactive educational resources as well, if more information about how they are being used becomes available.

The processing and logic involved in Learning Analytics can be used for other purposes different than education. The proposed LAS can help games in tasks like debugging the game design (statistics could show game points with no return), and testing (the LAS could spot if the user is playing the game as it was specified in the design).

Finally, it is important to note that the ultimate goal of Learning Analytics is to improve educational actions. We believe that learning analytics can help in establishing the educational value of games that use it. It is also important to take into account that monitoring students presents several ethics problems and privacy issues. Therefore transparency must guide all the design decisions.

A.6 References

[1] L. Communique, Towards the European higher education area: Responding to Challenges in a Globalised World. Techreport. 2007.

[2] T. Elias, Learning Analytics:definitions , processes and potential. Learning, 23. vol: 6, issue: 4, pp. 134-148. 2003

[3] M. F. Paulsen, Experiences with learning management systems in 113 European institutions. Educational Technology and Society. 2003.

[4] M. Brown, Learning analytics. Learning Circuits Retrieved March, vol. 2, issue 1, pp. 1-4. 2010.

[5] S. Williams and N. Williams, The business value of business intelligence. Intelligence, vol. 8 issue 301, pp. 30-39. The Data Warehouse Institute. 2003.

[6] M. Hassler, Web analytics. Redline Heidelberg, vol. 3, issue 1, pp. 1-14. 2010 doi:10.1080/19322900802660292

[7] S. Dawson, A. Bakharia, E. Heathcote, SNAPP : Realising the affordances of real-time SNA within networked learning environments. Learning. pp 125-133. 2010.

[8] J. Jovanovic, D. Gasevic, C. Brooks, V. Devedzic, M. Hatala, LOCO-Analyst: a Tool for raising teachers' awareness in online learning environments. The 2nd European Conference on Technology Enhanced Learning, Crete, Greece, 2007, pp. 112-126.

[9] K. Gilleade, A. Dix. Using frustration in the design of adaptive Videogames. Proceedings of the 2004 ACM SIGCHI International Conference on Advances in computer entertainment technology ACE 04 vol: 74, pp: 228-232. 2004

[10] V. Shute and J. M. Spector, SCORM 2.0 white paper: stealth assessment in virtual worlds. Learning. pp. 1-10. 2008

Appendix B

Tracing a little for big improvements: Application of Learning Analytics and Videogames for Student Assessment

Ángel Serrano-Laguna, Javier Torrente, Baltasar Fernández-Manjón, Pablo Moreno-Ger

Department of Artificial Intelligence and Software Engineering

Complutense University

Madrid, Spain

[aserrano, jtorrente]@e-ucm.es, [balta, pablom]@fdi.ucm.es

Abstract—Assessment is essential to establish the failure or success of any educational activity. Not only to measure the acquisition of the knowledge covered by the activity, but also to determine the effectiveness of the activity itself. The increasing adoption of new technologies is promoting the use of new types of activities in schools, like educational video games that in some cases are developed by the teachers themselves. In this kind of activity, interactivity increases compared to traditional activities (e.g. reading a document), which can be a powerful source of data to feed learning analytics systems that infer knowledge about the effectiveness of the educational process. In this paper, we discuss how a part of the students' assessment can be achieved semi-automatically by logging the interaction with educational video games. We conclude that even the

application of rather simple tracking techniques means an advantage compared to other systems that are fed with less quality data.

Keywords: Learning Analytics, educational games, data mining, assessment

B.1 Introduction

Increasingly, teachers of all education levels and knowledge branches are becoming enticed by the possibilities new technologies can offer to their daily work. Among other activities, teachers are starting to use educational videogames in order to explore new ways to educate their students [1], [2]. Although there is evidence to support that characteristics of videogames such as high interactivity, supply of engagement and challenge can improve the educational processes [3], [4], most teachers are reluctant to use them as evaluation tools and usually end up turning to traditional methods, like written exams. Videogames are eventually left as low-weight complements that have little or no impact on the final mark, even when games can support new assessment approaches [5], [6] as they foster problem-solving, critical thinking, observation and reasoning.

There are solid reasons that motivate this distrust. First, and probably most important, it is difficult to implement Question and answer structures with games. In games students are constantly solving problems at a certain pace that is designed to make the game challenging but not frustrating and the explicit introduction of a test or questionnaire in any form is seen as disruptive and breaks the immersive atmosphere [7]. Therefore evaluation through games can only be achieved by tracking students' interactions and extracting conclusions about the process. But to do so, teachers need tools that analyze data coming from the games and infer high-level knowledge that can be understood by a human (e.g. the student has problems with concept 'A' or skill 'B').

Most of commercial educational games used by educational institutions, developed by publishers or government education departments are distributed as black-boxes in order to protect intellectual property. As a consequence it is unfeasible to collect any kind of data

from them, leaving teachers with the only choice of using traditional methods, i.e. written exams.

Teachers doing their own games are using easy-to-use game editors [8]. These tools give them more control of the whole game, which allows them in some cases to extract information for evaluation purposes. However, to maximize effectiveness it is necessary that these tools include features to track students' interactions and display high-level results in a teacher friendly way without requiring technical knowledge.

Assessment systems of this kind can be complex to develop as they require advance techniques of data mining and precise information about the game and study domain. Nevertheless in this paper we discuss that even simple systems tracking a few types of traces can produce a lot of assessment information with little teacher intervention.

Our approach is framed in the Learning Analytics discipline, an emerging field [9] that advocates for the analysis of students' interaction data with online educational resources to better understand their learning process. This field is directly related to other disciplines that apply data mining processes, like Business Intelligence or Web Analytics. While Business Intelligence provides advice to take action on certain alerts by analyzing numeric data, Web Analytics is centered on the analysis of users interaction with web pages, and its main purpose is to report the collected information, but no conclusions are drawn automatically (data must be interpreted by the webmaster).

We propose a double approach. In first place we use a similar philosophy to Web Analytics: data collected from interaction is reported to teachers using convenient graphics. In this case data is treated in a game-independent manner and teachers are responsible to give meaning to those results, as no automatic inference is possible. In second instance we propose adding an extra layer that allows the automatic inference of conclusions by using game-specific data. In this case additional input from teachers is required, as they have to define assessment rules that help the system to infer conclusions.

This paper is structured as follows: first we present the types of traces that we pro-

pose to log to facilitate assessment, and what information can be extracted from them. Second, we discuss what kind of teacher-defined assessment rules are needed to reach a more fine-grained inference of conclusions and high-level generation of knowledge about the educational process. The last section discusses final remarks, restrictions and possible extensions.

B.2 Taces logged

Educational video games can be varied and embrace many genres. However, most games share a certain common characteristics and therefore a basic but fundamental set of interaction traces can be defined. The more concrete the game design is, the better these traces can be defined. However in this paper we elaborate on a general basis that could become the breeding ground for more elaborated models.

Our basic set of traces is presented below, with all the information associated that can be extracted from them.

B.2.1 Start game, end game, quit game

Generating a trace with a timestamp whenever a student begins to play some game provides information about when and who, if the students are uniquely identified, the game is being played. Adding a session identifier to the trace, we can track how many times any user tries to beat the game. With all this information we can also obtain group stats about the total number of users who played the game.

This information might seem trivial, but having the students uniquely identified is the first step towards automatic assessment.

Also, generating time stamped traces whenever a student finishes the game provides more relevant information. Firstly, this allows for determining if the student accomplished the game (i.e. s/he completed the main goal for which the game was designed). Secondly, by comparing the final and start timestamps the total time spent to fulfill the game can

be calculated. Global statistics can be generated as well, like students' success rate and mean completion times. Start traces are always generated but end traces can be logged only whether the student finishes the game. To deal with situations where the student quits the game it is necessary to generate another time stamped trace with information about the state of the game to let teachers know the exact point where students stopped playing. Sometimes, due to game platforms restrictions, the quit trace could not be generated. In those cases, the game state can be logged periodically at a fixed time rate. Checking the quit trace against the newest time stamped game state would provide information about the last game point reached. Moreover, by processing the evolution of these game states relevant information about the students' game play can be inferred.

B.2.2 Phase changes

Most of video games structure the narrative in chapters, missions or phases. Every phase can be associated with a secondary or mid-term goal in the game. A typical game structure is to consider the game as fulfilled or completed after all the secondary goals are fulfilled. In some games second term goals must be achieved in a specific order, while in other phases' can be explored freely.

If possible, logging traces with timestamps whenever a student starts and ends every phase provides relevant data. We obtain finer grained information about how the student is distributing her/his game play time within the games. Moreover, if the phase exploration sequence is not linear, teachers can gather insight on how each part of the game is being accessed.

Time spent in every phase can be used to identify most time-consuming parts of the game and lead to revising the game design if the results are unexpected (e.g. readjust difficulty, adapt content, etc.).

B.2.3 Significant variables

All games use variables to keep some sort of state. By tracking these variables the game play for every student can be reproduced.

A game can contain thousands of variables, but only a few are actually significant for assessment purposes. These variables can contain game scores or opportunities used to complete the game. A trace can be generated to reflect final values for these significant variables or, if necessary, when their values change.

This type of trace depends on the game mechanics and also on the possibility of working with game variables. In some game platforms this information is just kept hidden from the user/teacher.

B.2.4 User interaction

Previous traces are based on in-game situations and they contain information about what is going on in the game. But there is also interaction data produced by the user interaction with the game that may also contain relevant assessment information. Low level events such as mouse clicks, screen touches (on smartphones or tablets) and keys pressed register how students are interacting with the game and if all them were logged and reproduced, teachers could replay the whole game play of their students.

However, in most cases it results inefficient to analyze every single user interaction with the game. Then, it is necessary to filter interactions that are not relevant for assessment purposes. In our general approach, mouse clicks can be logged in order to create heatmaps marking hotspots distributed by phases or game scenarios for example. If the game control is based on keyboard interaction, pressed keys can be logged to check if students are using controls correctly.

When the educational videogame is multiplatform (having versions for desktop, mobile platforms, etc.) or have more than one input method (e.g. voice, mouse, keyboard, etc.) it may be necessary to have an abstract low level interaction model all different could be

mapped onto to enable comparison.

B.3 Extracting information

Once all data is collected from the game, we can start to extract some information from them.

B.3.1 Derived and combined data

The aforementioned types of traces can be automatically collected and displayed in human-readable reports to the teachers. The only requisite is to have a proper logging system available in the game and visualization tools to represent the data.

A game-independent analysis of data collected is useful, as it allows teachers to identify, for example, how long it took students to reach a certain point B from point A. However, knowledge can only be inferred automatically if information about the game is taken into account (e.g. structure, images, etc.). Using the previous example, if game-specific information is considered time intervals like those mentioned could be linked to the game structure to generate statistics about phase completion times.

Having game-specific information can be used to improve the quality of the reports as well. For example, heatmaps could use real images as backgrounds

Finally, combining different types of traces it is possible to obtain new information. For example, if in some phases users spent more time than expected, phase heatmaps could be checked in order to find what is the problem and redesign the game accordingly. This would allow an in-depth analysis of the feedback provided by the game by detecting students' misconceptions.

For example, using heatmaps it could be easy to identify situations where the player fails to interact with the right element or repeatedly interacts with an irrelevant character or object. These situations usually depict an incorrect or insufficient use of feedback, leading to an inappropriate level of guidance that prevents the student to reach an optimum decision

on what is the best action to progress in the game. By automatically identifying these situations, which is not a great challenge from a technical perspective, teachers or game designers would be able to provide more explicit feedback when needed.

B.3.2 Assessment

To generate knowledge about the process it is necessary to build a rule-based system on top of the interaction logging system. This system takes as inputs the assessment rules defined by the teacher and game-specific information. Useful information about the game includes its structure, specific goals, phases, and information about characters and objects present, for example. The system uses the teacher-defined rules to analyze the interaction logs and infer conclusions which are reported on high-level terms.

To avoid an excessive increase of the system complexity, assessment rules must be based on quantifiable parameters. Some of these rules are briefly discussed below:

Measuring times

Time spent to reach a specific some goal is key to determine student success. With the presented traces, it is possible to measure different time intervals: time spent to complete every phase and total completion time. Several assessment rules can be defined on these numbers. For example, a maximum time threshold can be set for each phase or the whole game, and use the comparison of the actual completion time with the threshold to automatically grade the student.

A minimum threshold could be set for games which have some repetitive actions that should keep students busy for a while. The maximum score would be given to students improving that threshold.

Variables values

All the significant variables chosen to be logged are a potential source of assessment information. These variables are usually numerical and assessment rules can be directly

defined on the comparison between actual and expected values of these variables.

For example, from a variable that represents a score it could be extracted a numeric mark based on its division by a maximum score. Or some goal can be considered as fulfilled only if a set of variables are greater than values defined by the teachers.

Group results

Also, complementary assessment rules can be defined by the teachers at the student group level.

Teachers could define minimum success rates for every game and program alarms for when these rates were not achieved. They also could define the number of students assumed for the game and be alerted if these number is not fulfilled when the deadline to complete the task is approaching.

By analyzing group results teachers could gauge the assessment system as well. For example, if the number of students that receive good marks is too little, time thresholds can be relaxed and get new assessment reports automatically.

B.4 Final remarks

Some considerations about the model proposed deserve discussion. The game platform used for running the games must comply with some technical requisites. First, it must allow collecting or generating traces as described in this paper. This issue could be addressed extending open source game engines or using one with a built-in tracking system. Second, the game platform must include an explicit model to represent the definition of a game. This model should be kept in separate files from the code that runs the games in a format that is easy to process by a machine. And third, data logged must be stored somewhere, normally in a database, which could increase the difficulty to deploy the games. But the game engine could also log locally (in files with some specified format, for example) and then pass the results to an external analyzer. For data visualization, one of the many open

source libraries available could be used to generate graphics and reports.

In this paper we propose a double-step approach that applies Learning Analytics techniques in educational videogames. In first place it pretends to be general enough so to be applied to multiple game genres and game mechanics, treating information collected as game-independent to produce reports that teachers could analyze. In a second step, an additional assessment layer could be used to generate more depurated reports. These reports serve two purposes: assess and evaluate how students' progress within the game, and also debug the game design, spotting weak points in the game that could be fixed by the teacher if necessary.

We think it is a first step towards a new model of student assessment based on educational games that can complement other methods. These ideas will not substitute traditional assessment techniques, but they can provide more information about the educational process to the teachers in a rather automatic way. More sophisticated algorithms can be developed in order to expand and obtain finer-grained information, narrowing the games genre or the game mechanics as needed.

B.5 References

[1] L. Johnson, S. Adams, and M. Cummins, NMC Horizon Report: 2012 K-12 Edition. Austin, Texas: The New Media Consortium, 2012, p. 44.

[2] L. Johnson, S. Adams, and M. Cummins, NMC Horizon Report: 2012 Higher Education Edition. Austin, Texas: The New Media Consortium, 2012.

[3] L. A. Annetta, J. Minogue, S. Y. Holmes, and M. Cheng, "Investigating the impact of video games on high school students' engagement and learning about genetics," Computers and Education, vol. 53, pp. 74-85, 2009.

[4] H. Tuzun, M. Yilmazsoylu, T. Karakus, Y. Inal, and G. Kizilkaya, "The effects of computer games on primary school students' achievement and motivation in geography learning," Computers and Education, vol. 52, no. 1, pp. 68-77, Jan. 2009.

[5] A. Serrano, E. J. Marchiori, A. del Blanco, J. Torrente, and B. Fernandez-Manjon, "A framework to improve evaluation in educational games," in Proceedings of the 2012 IEEE Global Engineering Education Conference (EDUCON), 2012, pp. 1-8.

[6] Á. del Blanco, J. Torrente, E. J. Marchiori, I. Martínez-Ortiz, P. Moreno-Ger, and B. Fernández-Manjón, "A framework for simplifying educator tasks related to the integration of games in the learning flow," Education Technology and Society.

[7] J. Chen, "Flow in games," Communications of the ACM, vol. 50, no. 4, pp. 31-34, 2007.

[8] J. Torrente, Á. Del Blanco, E. J. Marchiori, P. Moreno-Ger, and B. Fernández-Manjón, "<e-Adventure>: Introducing Educational Games in the Learning Process," in IEEE Education Engineering (EDUCON) 2010 Conference, 2010, pp. 1121-1126.

[9] R. Ferguson, "The State of Learning Analytics in 2012: A Review and Future Challenges a review and future challenges," Media, no. March, 2012.