A General Tracking and Auditing Architecture for the OpenACS framework

Jorge Couchet[1], Olga C. Santos[2], Emmanuelle Raffenne[2], Jesús G. Boticario[2], and Daniel Manrique[3]

[1]Applied Artificial Intelligence Center, Computer Science School. ORT, Montevideo, Uruguay
jorge.couchet@universidad.ort.edu.uy
[2]aDeNu Research Group, Artificial Intelligence Department, Computer Science School. UNED, Madrid, Spain
{ocsantos, eraffenne, jgb}@dia.uned.es
[3]LIA Research Group, Artificial Intelligence Department, Computer Science School. UPM, Madrid, Spain
dmanrique@fi.upm.es

Abstract. The paper describes the Tracking and Auditing Engines (TAE) in process of development for the OpenACS framework through the implementation of a tracking subsystem and an auditing API built upon it. The main theoretical considerations that must fulfill such system are discussed in the paper, specially the differences between the responsibilities and functions for the tracking and auditing processes. The data required and where to get it from the framework, the architecture designed, and the technology to be used in the implementation are also presented. As a practical use of the TAE, the paper presents on-going authors' research that is based on analyzing dotLRN users' interactions. These research works will benefit from the audit trails provided by the TAE.

Keywords: Tracking, auditing, web development framework, OpenACS, dotLRN, web service support.

1 Introduction

The Open Architecture Community System (OpenACS) is a full featured web development framework used by many big players in several important social areas, such as the NGOs world as their infrastructure platform (Greenpeace, AIESEC, and others). dotLRN, an application for e-learning built on top of OpenACS framework is also widely used (University of Heidelberg, the MIT Sloan School of Management, Spanish National University for Distance Education, and many others). In those contexts, a key feature is the tracking and auditing capabilities. It should be compliant with the standards required in each case and facilitate the understanding of the software system (or subsystem) behavior. In particular, it should i) help to evaluate the adequacy of the policies, procedures and other mechanisms implemented, ii) provide an on-going feedback to the administrators, iii) assess the system for security, and iv) support the development of new useful applications over the audit trails.

Despite the importance of having a coherent common data model and API to gain access to the tracking and auditing services, the support provided in the OpenACS framework is rather limited. There exist services to solve specific situations, but the framework lacks of a generalized and flexible architecture that can be used to satisfy the different needs that may arise. OpenACS offers two ad-hoc solutions: the *ClickStream* [1], [2] and the *User Tracking* [3] packages, both of them with the aim to resolve specific needs and problems. Those packages are very good at resolving the concrete situation for which they were designed, but they do not offer a metadata model to solve any situation where the tracking and auditing services may be required.

This paper presents a general tracking and auditing architecture that implements a metadata model that intends to extend OpenACS functionality. First, the tracking and auditing basis are introduced. Next, the details of the proposed architecture are presented. Then, the on-going development of a new application built over the audit trails, is shown. Finally, the benefits of the proposal and its open issues are discussed.

2 Tracking and Auditing basis

The Tracking and Auditing Engines (TAE) of a software system as is described in the Fig. 1, provide the means to *record* all the *actions* performed both by the direct *users* of a system and by the system itself; where an *action* is defined as a specific piece of *functionality* of the system. The execution of an *action* implies some kind of processing to be applied over an *object*, or a set of *objects* on a *dataset*. An *object* in this context is defined as a set of *attributes*, where an *attribute* is a unique name-value pair [4]. We are not referring here to OpenACS objects.

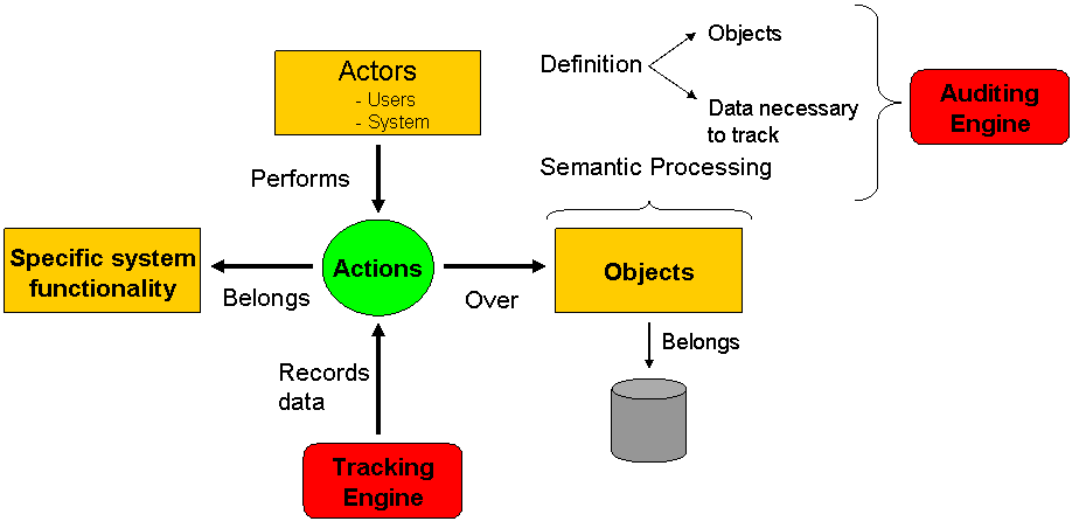

Fig. 1. Tracking and auditing processes

It is important to separate the responsibilities of the Tracking and the Auditing processes, as is specified in Fig.1. The Auditing Engine is in charge of the *definition* of the objects that need to be audited, the *definition* of the necessary data to be recorded in order to audit the defined objects, and the performing of the *semantic processing* over the recorded data, in order to extract the objects and the meaning of the interaction between those objects. The Tracking Engine is in charge of *recording* the data defined by the Auditing Engine.

2.1 The Tracking and Auditing Engines requirements

The requirements that a TAE should fulfill could be classified in two types, *functionality* and *security* requirements [4]. The first one implies that the TAE should implement the following minimum requirements: *a)* record the actions performed through the system, *b)* record the object states, *c)* provide means to retrieve audit data, *d)* provide means to track the history of an object, and *e)* provide means to detect any external change of the data.

The security requirements to be addressed are: a) the audit data must be stored in a secure manner, b) continuous audit must be assured, and c) compliancy and integration with standards must be provided. Those requirements can be analyzed and assigned to each TAE module, according to the responsibility described above.

2.1.1 Tracking Engine requirements

The Tracking Engine should fulfill the following *functional* requirements:

- Record the actions performed through the system: in a consistent way and in a universal format.

The Tracking Engine should fulfill the following *security* requirements:

- The audit data must be stored in a secure manner: only authorized users can access the stored information, and no external modification of the data must be allowed.
- Continuous audit must be assured: during the whole life cycle of the application objects a full audit must be maintained.
- Compliancy and integration with standards must be provided: the Tracking Engine should fulfill the standards required, and provide the means to help the applications in the framework to fulfill the standards required.

2.1.2 Auditing Engine requirements

The Auditing Engine should fulfill the following *functional* requirements:

- Record the object states: keep track of the initial object's state (before the action), and the final object's state (after the action).
- Provide means to retrieve audit data: a reporting engine that enables to the authorized users to query the audit data.
- Provide means to track the history of an object: the reporting engine must be capable to search the history of an object.
- Provide means to detect any external change of the data: detect direct (bypassing the framework) modifications to the dataset.

The Auditing Engine should fulfill the following *security* requirements:

- The audit data must be stored in a secure manner: only authorized users can access the reports and Auditing Engine configuration.
- Compliancy and integration with standards: the Auditing Engine must complement the Tracking Engine in this task.

2.2 Users of the TAE

The users of the TAE can be divided into two groups [5]. The first group consists of the *auditor*, who is an individual with administrative duties. The auditor selects the events to be audited on the system, configures the system, and analyzes the trail of audit events.

The second group of users of the audit mechanism consists of the system users themselves. This group includes the administrators, the operators, the system programmers, and all other users. They are considered users of the audit mechanism not only because they, and their applications, generate audit events, but because they must understand the audit mechanism and the impact that it has on them.

3 TAE proposed Architecture for OpenACS

The main goal of the TAE architecture is to offer a coherent common data model and API that the applications can use in order to access to the tracking and auditing services. With the proposed architecture, a new application only needs to hook to the metadata model offered in order to gain access to the TAE services, avoiding or minimizing the amount of code repetition.

3.1 OpenACS data gathering

When a user interacts with the OpenACS framework, he goes through the AOLserver web server, so all the user performed *actions*[1] are going through the AOLserver. For this reason, the AOLserver's *filters* have been selected as the main data gatherer for the TAE system.

[1] The exception is a direct modification of the OpenACS database.

3.2 Architecture Description

The main features for the proposed *General Tracking and Auditing Architecture for the OpenACS Framework* can be observed in Fig. 2.

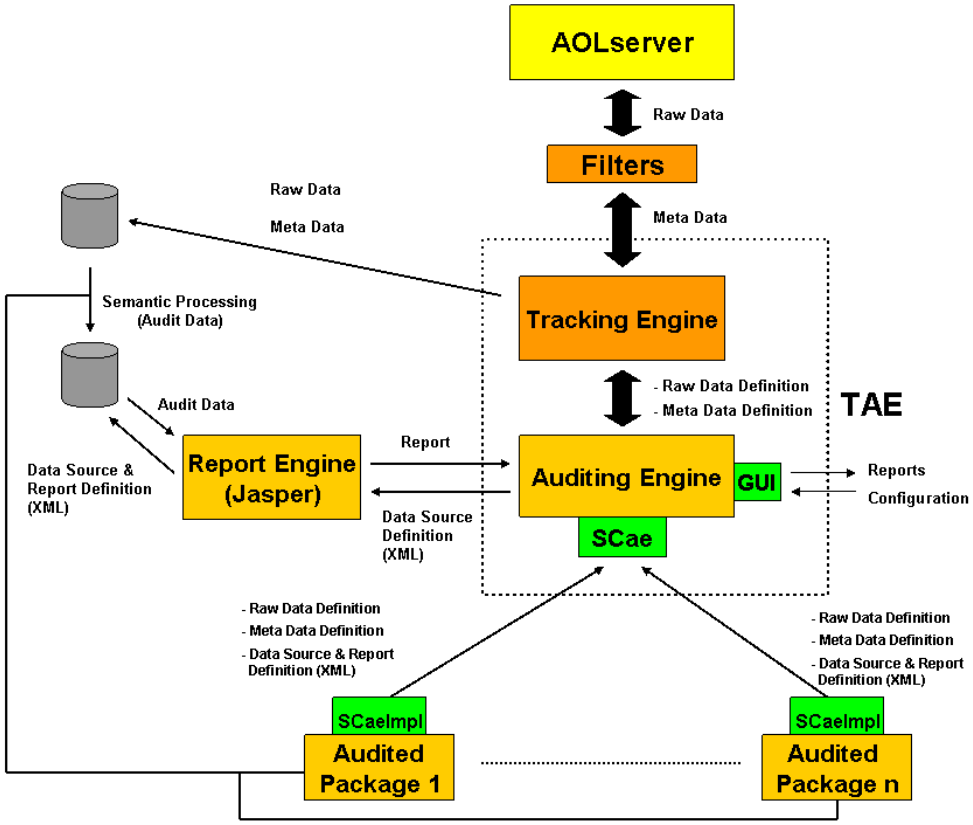

Fig. 2. General Tracking and Auditing Architecture for the OpenACS Framework

The architecture's components are:

- *Auditing Engine*: It offers a centralized access and administration to the TAE, and defines a *Service Contract* for the Auditing Engine (*SCae*) in order to distribute the audit function among the different OpenACS packages, each one with its own information.
- *Tracking Engine*: It hides to the Auditing Engine (and the TAE user) the recording mechanism. The engine receives the definition of the data (raw data and metadata) that needs to be recorded, and creates automatically the AOLserver filters needed.
- *Report Engine*: It hides to the Auditing Engine (and the TAE user) the complexities of the report creation. The engine receives the data source definition (XML) and the report configuration (XML), and creates the requested report.
- *SCae Implementation for the audited packages*: Each OpenACS package that would like to take advantage of the Tracking and Auditing functionality has to provide its implementation for the SCae. It should define the *objects* of interest, and the *actions* that need to be recorded, providing to the Auditing Engine their definition. It must also provide its own *semantic processing* functions in order to do the meaning extract from the Tracking Engine's recorded data.

4 Applications

4.1 Previous Background

An important part of the authors' research is to develop new machine learning algorithms in order to be used to extract useful knowledge from big datasets. For example, the logs of a web framework that supports an online community. This research work has produced a Java-based machine learning library that automatically captures, through the application of an adaptive outlier detection algorithm (LOF), the emergent properties of self-organizing system of neurons, and generates a 3D visualization of them [6].

Self organization is the ability of a system to adapt its internal structure to the external stimulus received though its sensors. The adaptation process should satisfied the following conditions; *a)* no external intervention (unsupervised learning), and b) the internal structure of the system represents the structure of the external data space that is relevant to the system. A kind of self-organizing system are the self-organizing maps (SOM), which are neural networks (a system of neurons), trained (the adaptation process) using an unsupervised learning algorithm (the *a* condition), to produce a low-dimensional, discretized representation of the input space of the training samples, called a map (the *b* condition). The map seeks to preserve the topological properties of the input space, and therefore de development of visualization techniques over it, is useful for visualizing high-dimensional data [7].

Emergence appears in a system through the properties of the collective behavior of the elements which the system is composed of. Looking at the system as a whole, a global emergent property comes as a new entity of the interdependency of its parts. The emergent property is the ability of a system to create a new entity at a higher level. This level change is the product of the cooperation of a large number of elemental processes. The emergent structures created give a more abstract description (higher level) of the original complex system of elemental processes (lower level) [8]. Emergence through self organization is a non trivial property of SOMs. For emergence to appear, a system should consist of simple but highly correlated elements. Without these correlations, emergence is not possible. A SOM with large numbers of neurons, being large of the order of thousands, presents emergent properties, and this kind of SOMs are named Emergent SOMs (ESOM) [9].

The Local Outlier Factor (LOF) algorithm finds data points (outliers), that lies an abnormal distance from other values in a multidimensional dataset [10]. A LOF value is defined for each object in the dataset being studied. It is called local since only a restricted neighborhood of the object is used to compute it. This approach makes it possible to adapt dynamically in the presence of clusters with different densities as opposed to other global methods. To compute the LOF, the parameter k has to be chosen, which is taken as the number of nearest neighbors used to define the local neighborhood of an object. For each object, its LOF is not a binary value. Instead, it represents the degree by which that object can be considered an outlier. A theoretical property is that a LOF value close to 1 means the object is deep inside the cluster.

The machine learning algorithms developed by the authors extends the ESOM concept, to the recurrent ESOM with LOF (LR-ESOM); which adds to the ESOM an explicit context representation, and modifies its distance function and learning algorithms with the goal to deal with sequences, through a fractal codification of them, obtaining a recurrent ESOM (R-ESOM). Finally, over the R-ESOM map, the LOF algorithm is used to capture the map's emergent properties automatically (LR-ESOM). Additionally, a 3D visual tool was added for the LR-ESOM map, through the development of the LMATRIX concept [6]. The 3D representation of the L-Matrix assigns coordinates (x, y, z) to each unit, where (x, y) are the position of the unit in the map and z its LOF. The cloud of points obtained is interpolated using Bi-Cubic Bezier Patches [11] to generate a smooth surface. In order to take advantage of the LOF theoretical properties, the surface is colored, assigning the same color to units with LOF inside [0,LINF], with LINF = 1. The color for units with LOF higher than LINF is calculated using a mapping between a color scale and the range (LINF, LMAX], where LMAX is the maximum LOF of the units. Changing LINF may be used as a zoom in/out function of the emergent properties.

4.2 Mining the dotLRN User's Needs

A Tracking and Auditing architecture for managing the interaction data of the system that supports standards and metadata provides a very powerful mechanism to develop adaptive functionality for the end-

users. As a practical use of the OpenACS's TAE, the authors are developing a new model which rely on this processing layer, utilizing the properties of auto-organized systems and combining them in learning scenarios defined in terms of educational specifications, such as IMS Learning Design (IMS-LD) [12]. IMS-LD provides a workflow oriented approach for defining the sequence of activities users may perform in an e-learning scenario. Research works as aLFanet project (IST-2001-33288) have already used the IMS-LD design, together with an intensive metadata description of the users and the contents, to manage and support runtime adaptation [13]. By combining both approaches (i.e. auto-organized systems and learning design specifications), we intend to mine, analyze and categorize the sequence of actions followed by learners in learning environments to provide dynamic recommendations that improve the efficiency and efficacy of their learning. This dynamic support for recommending users in OpenACS is also being implemented [14].

The model proposed is built on a sequential and independent three-level process (see in Fig. 3).

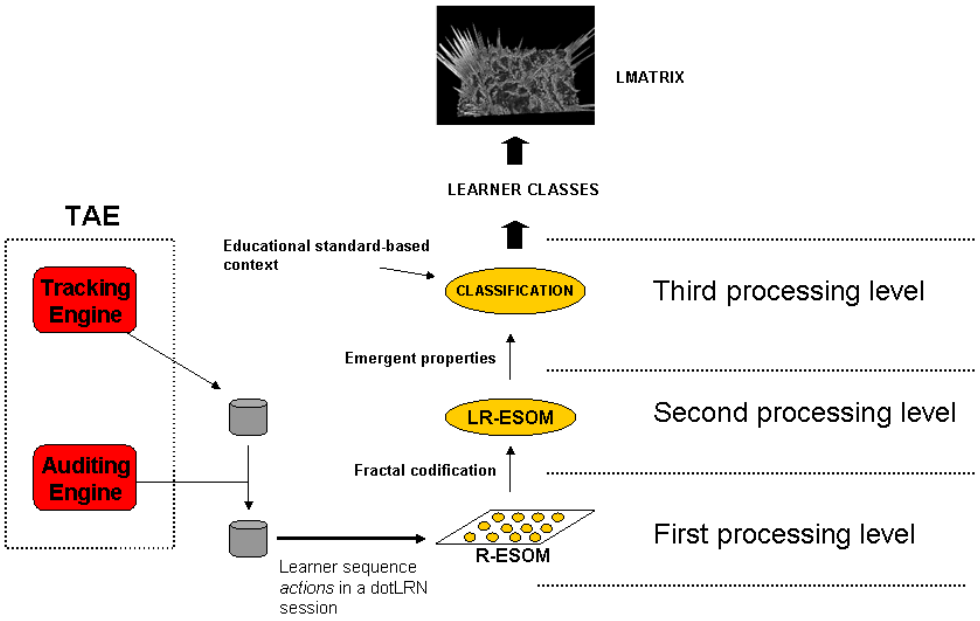

Fig. 3. Model to mine the dotLRN user's needs built over the TAE's audit data

From the concepts and the machine learning library detailed in the preceding section, the first level of the model is obtained following an auto-organized approach with a Recurrent Emergent Self-Organizing Map (R-ESOM). At this level, a fractal codification of the sequence of actions done by the learner in a certain interval (dotLRN session) is made. With this fractal codification emergent global properties are obtained. These properties arise from the collective behavior and interrelation of the different dotLRN users action sequences. Those emergent global properties are automatically extracted with the LOF algorithm and are viewed as the different regions in the new LR-ESOM map. Finally, an educational standard-based context is given to the results obtained in the previous levels, so that it can be applied in standard-based learning scenarios. This third level categorizes learners in classes according to their individual learning needs. This innovative modeling approach uses the visual method LMATRIX, that allows the 3D generation of visual patterns for the analysis and interpretation of learner's sequence of actions in the environment.

5 Conclusions and Future Work

The TAE architecture proposed intends to resolve an open issue in the OpenACS framework, which is the lacking of a proper TAE subsystem. This proposal offers a metadata model fully open and very flexible, in which different modules (the audited packages) and their corresponding data models can be added or

removed, depending of the diverse audit needs that may arise. The implementation of a TAE subsystem with the characteristics described in the current paper can lead the OpenACS framework to the next maturity level. Offering a tracking and auditing functionality embedded in its kernel will make it stand out other systems and stress its role in the collaboration and online software communities, extending also its penetration in that market.

In order to assure a scalable TAE subsystem, there are still some open issues to solve. One is to determine the best mechanism for the recorded data growth. For this problem, there could be two solutions. The first one is to put a limit on the number of changes that will be tracked on each record. The second involves setting a periodic process that archives the log data to a separate file and deletes the archived content from the log files.

The other open issue is the record process itself. More specifically, it is necessary to determine if the database is used directly to store the tracked data, or if it is better to use an intermediate step, which involves storing first the data in a plain text file, and next use a background process that transfers the data to the database.

Future work is focused on implementing the TAE described here. In the design process of the architecture we have dealt with a very challenging question. And this question is how to extend the current OpenACS object API to support directly auditing functions, while the API is endowed with the necessary knowledge about the meaning of the interaction between objects.

References

1. Carroll, N. Clickstream Data Warehouse. Undergraduate Thesis Projects 2002, http://www.weg.ee.usyd.edu.au/projects/ug_projects#thesis2002
2. Surath, Vasudev.: A Web Traffic Analysis Tool. Undergraduate Thesis Projects 2003, http://www.weg.ee.usyd.edu.au/projects/ug_projects#thesis2003
3. Ortega, D., Arozarena, P., Hernández, R.: Desarrollo de un Paquete para el Seguimiento de Usuarios para dotLRN, http://dotlrn.org/file-storage/view/madrid05/09.pdf
4. Patriciu, V., Vaduva, C., Morariu, O., Vanca, M., Tofan, O.: Modeling the Audit in IT Distributed Applications. J. Applied Quantitative Methods. 2, 109—107 (2007)
5. A Guide to Understanding Audit in Trusted Systems, http://www.fas.org/irp/nsa/rainbow/tg001.htm
6. Couchet, J., Ferreira, E., Fonseca, A., Manrique, D.. A Novel Architecture for the Classification and Visualization of Sequential data. Springer Verlag, pp 730-738 (2007)
7. Kohonen, T.: Self Organizing Maps. Third edn. Springer Verlag (2001)
8. Yaneer, B.: Dynamics of Complex Systems. Addison-Wesley (1997)
9. Ultsch, A.: Emergence in Self-Organizing Feature Maps, In Proceedings Workshop on Self-Organizing Maps (WSOM '07), Bielefeld, Germany, ISBN: 978-3-00-022473-7
10. Breunig, M.M., Kriegel, H.P., Ng, R.T., Sander, J.: LOF: identifying density-based local outliers. SIGMOD Rec. 29(2) 93–104 (2000)
11. Salomon, D.: Curves and Surfaces for Computer Graphics. Springer (2005)
12. IMS Learning Design Specification, http://www.imsglobal.org/learningdesign/
13. Boticario, J.G. and Santos, O.C. An open IMS-based user modelling approach for developing adaptive learning management systems. Journal of Interactive Media in Education. Special issue on Adaptation and IMS Learning Design, 2007
14. Santos, O.C., Raffenne, E, Granado, J. and Boticario, J.G. Dynamic support in OpenACS/dotLRN: Technological infrastructure for providing dynamic recommendations for all in open and standard-based LMS. Proceedings of the International Conference and Workshops on Community based environments, 2008 (in press).