Review of current student-monitoring techniques used in elearning-focused recommender systems and learning analytics. The Experience API & LIME model case study

Alberto Corbi, Daniel Burgos

UNIR Research,Universidad Internacional de La Rioja - UNIR

***Abstract* — Recommender systems require input information in order to properly operate and deliver content or behaviour suggestions to end users. eLearning scenarios are no exception. Users are current students and recommendations can be built upon paths (both formal and informal), relationships, behaviours, friends, followers, actions, grades, tutor interaction, etc. A recommender system must somehow retrieve, categorize and work with all these details. There are several ways to do so: from raw and inelegant database access to more curated web APIs or even via HTML scrapping. New server-centric user-action logging and monitoring standard technologies have been presented in past years by several groups, organizations and standard bodies. The Experience API (xAPI), detailed in this article, is one of these. In the first part of this paper we analyse current learner-monitoring techniques as an initialization phase for eLearning recommender systems. We next review standardization efforts in this area; finally, we focus on xAPI and the potential interaction with the LIME model, which will be also summarized below.**

***Keywords* — LIME model, eLearning, Conceptual Educational Model, Rule-based recommender system, Informal learning, Social interaction, Learning Tool Interoperability, User monitoring**

I. INTRODUCTION: REVIEW OF RECOMMENDER ENGINES, ELEARNING AND NEED FOR USER INPUT DATA

RECOMMENDER engines deliver suggestions based on collected information on preferences, general user behaviour and even items bought or content searched. Trendy online stores and services massively apply this approach ([HYPERLINK \l "1167344" 1]). The information can be obtained explicitly (by processing users' manual tiering) or implicitly, typically by monitoring users' behaviour, such as songs downloaded, applications launched, chat transcriptions, web sites visited, PDFs read, or ebooks transmitted to ePub readers (2]).

Recommenders can also make use of demographic info and social information (e.g., followers, e-friends, posts, replies, chat rooms, and others), as well as geographical location data or even health signals (e.g., pedometers, blood pressure).

Collaborative filters ([HYPERLINK \l "marlinmodeling" 3]) are very often used by recommender systems along with content-, knowledge- and social-based filters. Implementation of these filters has grown as access to the Internet has become more widespread in recent years. They can be used for any type of reachable media (e.g., movies, music, television, and books) and in many different scenarios, such as eLearning, e-commerce, mobile applications, search, dating, etc. These filters need to access as much of a user's navigation and behaviour history as possible in order to offer fine-tuned purchase options or action tips.

Memory-based methods use similarities and ratings from all users who have manually expressed their preferences/level of satisfaction on a given object/issue. These similarities represent the distance between two users and their tiered records. Model-based methods establish first the sets of similar users by using Bayesian classifiers, neural networks and fuzzy systems. Generally, commercial recommender engines use memory-based methods. On the other hand, model-based methods are usually associated with research environments, including eLearning. Hybrid techniques can also be applied and have been demonstrated to be of much importance to assist and guide users through systems. Hybrid recommenders merge different types of techniques in order to get the most out of each of them. Finally, we have rule-based recommenders, like LIME, which will be analysed below. In rule-based systems, a set of conditional filters are manually defined and triggered when necessary in order to deliver the appropriate recommendation to the user/learner.

The increase in the attention paid by the research community to recommender systems is striking, as has already been pointed out in 4]. Figure 1 shows, on the Y-axis, the number of cited papers from each year as of 2013. The size of each bubble corresponds to the number of proceeding articles for that given year. It can be noted that there is a peak of interest around 2009.

DOI: 10.9781/ijimai.2014.276

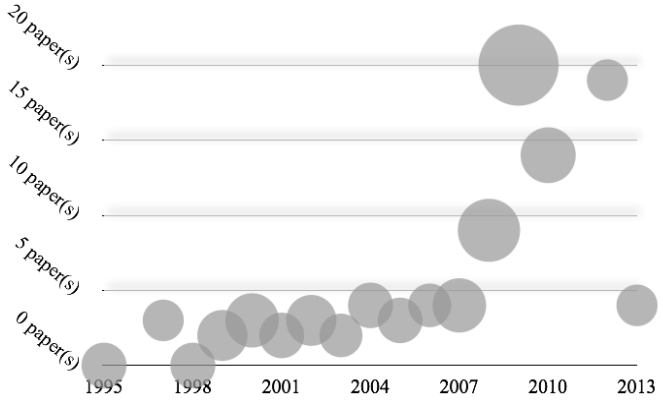


Fig. 1. # of papers and workshops related to subject *recommendation*

With the development of sophisticated eLearning environments and Learning Management Systems (LMS) ([HYPERLINK \l "abedour" 5]), *personalization* is also becoming an important feature. Personalized learning occurs when eLearning platforms are designed according to educational experiences that fit the needs, goals, and interests of each individual learner. Personalization can be achieved using different recommendation techniques, very similar to those just summarized. Ideally, recommender systems in eLearning environments should assist students in finding relevant learning actions and materials that perfectly match their profile and the best way towards self-education. The right time, the right context, and the right way are also critical. Recommenders should also keep learners motivated and enable them to complete their academic activities in an effective and efficient way. Personalization should take place, not only on enrolment-limited online campuses or *Small Private Online Courses* (site courses, college classes, student groups, etc.), but also on the now trendy MOOCs: *Massive Open Online Courses* environments (6], [HYPERLINK \l "mooeurope" 7]), where enrolment rate can be up to a few thousand students. In other words, a recommender system should have the ability to efficiently scale up or down independently of the number of students and without losing sight of the goal of improving individualized education.

Recommender systems (especially in eLearning) can also suffer from the *cold-start* problem. Cold start occurs when there is an initial lack of input data (ratings, logged actions from users, etc.) to trigger or initialize the appropriate algorithm. We can distinguish two main kinds of cold-start variants: *new item* and *new user* ([4]). The new-item problem arises because new items entered do not have initial ratings/inputs from users. Also, a priori, *new users* in a system might not yet have provided any input info, and therefore cannot receive any personalized recommendations.

Independently of the algorithm used, the identifiable potential issues (like cold start) and the scenario of application, recommender systems require input data in order to behave properly (8]). This data can be manually entered ([HYPERLINK \l "Bobadilla20111310" 9]) by the user (ratings, explicit opinions, etc.) or implicitly obtained by monitoring software. In an eLearning environment, the latter approach is more likely to be the chosen one.

We now list the most common techniques used for monitoring learners' actions in an LMS. The next sections will present the Experience API and other standardization efforts as new and modern ways of logging learner actions, chosen materials, student paths, etc., and serving them to recommender systems. Finally, we introduce the rule-based LIME model and discuss how can it be fed from an Experience API Learning Record Store repository (which we will also discuss) in order to properly operate and deliver rule-based recommendations to students.

II. Basic System-Dependent monitoring techniques

There exist three main different non-standard ways of interacting with Learning Management Systems (and electronic systems in general) and extracting user/learner data (also summarized in Figure 2):

A. Web Services

The first and most immediate way to obtain learner input data is through LMS-dependent web services and API calls. Modern LMS (10]) do usually offer simple, elegant, industry-standard and compelling ways (WSDL, SOAP, RPC and REST) of accessing their internal information and retrieving needed data. This approach has one main drawback: not every service needed is implemented and/or enabled by default. This could be easily tackled if we are granted access to the LMS infrastructure in order to add these missing *sockets* or activate existing *disabled-by-default* ones. However, this is not always possible in many scenarios (e.g., proprietary cloud-based campus environments). Another clear disadvantage is that developed web services are very unlikely to be compatible between two distinct LMS, making it necessary to re-code each of them for every platform and software version.

B. Scrapping

Web scrapping consists of, on the one hand, running automated HTTP(S) requests that retrieve the same pages and HTML documents as a user would fetch by operating a web browser manually ([HYPERLINK \l "6112910" 11]). On the other hand, after such requests have succeeded, data can be distilled, examined and applied to some sort of scripting/analytics. Most HTTP command line (CLI) client programs/libraries allow authentication and form submission, which is usually enough for most purposes. Although web scrapping seems the most compatible form of mechanized data-mining, we still face a minor problem: some LMS make huge use of Javascript for accessing resources and building routes to them. In this scenario, CLI web clients are not enough and should be superseded by what are known as headless web browsers, explained in previous studies (12], [HYPERLINK \l "Grigalis:jucs_20_2:unsupervised_structu" 13]). Such browsers are scriptable, run without any user interface, and best of all understand and can execute Javascript code without user intervention.

The result of a scrapping operation is usually an HTML file or a set of files of this kind, which should be processed afterwards (14]) in order to extract the desired monitoring information. As HTML is a descendant of XML, any XML parsing technique (XPath, XQuery, XSLT, etc.) and technology applies here, e.g., Nokogiri ([HYPERLINK \l "Hun13" 15]).

C. Raw database access

This is by far the most-often-seen method in the literature, which implies direct access to the system database. This approach has several advantages and disadvantages. The main advantage is speed, since no intermediaries, software layers or no other different APIs play a role in data retrieval (apart from the SQL engine and the APIs themselves). The most significant downside is possible database scheme migrations and incompatibilities as new versions of the server software are deployed.

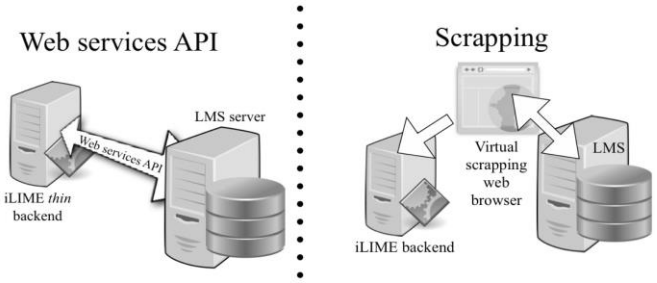


Fig. 2. Basic web monitoring techniques

These three techniques spring into action at some point or another of the student data-mining process, in different ways and with different goals. Some monitoring models, as seen in 16] and [HYPERLINK \l "Mazza04gismo:a" 17], make use of reports and logs derived from data contained in server temporary files. Other research efforts, such as the one presented in 18], have used closed-systems and setups, with their own specific monitoring methods and engines. The study presented in [HYPERLINK \l "5561329" 19] makes use of quiz results as input for a research recommender model. The authors of 20] and [HYPERLINK \l "6033004" 21] feed their recommenders with web-browsing behaviour. In22], the authors gain direct access to a Moodle instance database in order to boot their *Predictive a priori algorithm*. The model in [HYPERLINK \l "ElB10" 23] initially presents students with a test to identify his/her personality in Myers-Briggs dimensions. The authors in 24] suggest obtaining input data not only from the server and client sides, but also from proxy servers. In [HYPERLINK \l "Kardan:2012aa" 25], content recommendation needs each student to self-monitor him/herself: learners estimate different indexes themselves and compare them with actual values, which are retrieved by the system. The model presented in 26] uses the *AprioriAll* algorithm to immediately build sequences from server logs, which are used in conjunction with tags in order to deliver recommendations. The model in [HYPERLINK \l "conf/wec/WangH05" 27] also makes use of the AprioriAll algorithm using only web logs. In 28], again, only web-browsing activities of learners are monitored, but these are then subdivided into *web content mining*, *web structure mining* and *web usage mining* realms.

We also find learning research software prototypes, like the PSLC Datashop initiative from the Pittsburgh Science of Learning Center [HYPERLINK \l "Stamper:2011:MED:2026506.2026609" 29], which has defined its own XML DTD schema as a logging scaffold for their *Tutor* learning research platform. Some approaches rather build a dedicated tool or patch applied to a LMS, as in 30] with the MOCLog project for Moodle.

The Experience API and other standardization proposals for the monitoring phase, presented below, advocate a completely new and cohesive approach to this critical phase in the recommendation/learning analytics workflow.

II. Standard Specifications for Monitoring

The aforementioned non-standardized approaches to user/learner monitoring can be applied on fully controlled scenarios and research projects. However, they turn out to be unsatisfactory in real academic environments managed by third-party institutions.

There exist a few proposals that aim at standardizing the monitoring and logging of user actions. Almost all are based on the *Resource Description Framework*, or RDF [HYPERLINK \l "Pan09" 31]. The idea behind RDF is something called the *triple*. A triple can really be condensed to a plain sentence structure:

- subject
- phrase that characterizes a relationship
- object.

Example: Daniel – *is the author of* – this paper.

Triples are extremely useful and simple, and provide a grammar for the so-called *semantic web*.

Also, some of these specifications include some sort of software and database back-end service, linked APIs and query language that allow learning platforms to send and store monitoring data and third-party learning analytics software to query and retrieve analysable data. We summarize here the most important and paradigmatic monitoring specs:

The **Caliper** framework/**Sensor** API was proposed by the IMS Global Consortium and follows the triple metaphor. It is built around the following concepts (32]): *Learning Metric Profiles* that provide an activity-centric focus to standardize actions and related context; *Learning Sensor API* and *Learning Events*, which drive tools and an associated analytics service solution; and finally, Learning Tool Interoperability (LTI), which enhances and integrates standardized learning measurements with tool interoperability.

IEEE 1484.11.1/IEEE 1484.11.2 ([HYPERLINK \l "IEE05" 33]) provides a complex data model structure for tracking information on student interactions with learning content. Additionally, an API allows digital educational

content coming from the LMS and third-party services to query and share collected information.

JSON Activity Streams (34]) is the name of the specification published by IBM, Google, MySpace, Facebook, VMware and Microsoft. Its goal is to provide sufficient metadata about an activity such that a consumer of the data can present them to a user in a rich human-friendly format. It does not provide a logging service, just the specification of the message format.

Finally, we also have the **Experience API**, which will be addressed in the next section.

Security and privacy models can also be applied in all specs cited above. Network communications can be encrypted and the subject can be anything but the learner's real name. Learning analytics researchers and logging storage implementers are responsible for the ethical usage of the compiled info coming from student monitoring. As with any other area related to digital mining, trust, accountability and transparency must always prevail ([HYPERLINK \l "Par14" 35]).

III. The Experience API Specification

The Experience API (or *xAPI* for short) is an eLearning monitoring specification developed by Rustici Software and the Advanced Distributed Learning Initiative (ADL), and is aimed at defining a data model for logging data about students' learning paths (36]). It also furnishes an API for sharing these data between remote systems, as we will see later. The Experience API allows, among other things, the tracking of games and simulations, real-world behaviour, learning paths and academic achievements. xAPI defines independent mechanisms, protocols, specifications, agreements and software tools for monitoring any imaginable scenario (Figure 3): from online campuses and student behaviour to workforce control ([HYPERLINK \l "6530268" 37]).

Fig. 3. Examples of usage of the Experience API

xAPI also uses JSON to transfer states/sentences to a central web service. This web service allows clients to read and write data in the form of *sentence objects* that share the foundations of the aforementioned triple scheme. In their simplest conception, sentences are in the form of *actor*, *verb* and *object/activity*, like the examples in Figure 4. A JSON xAPI message could resemble the following:

```
{"id": "3f2ef28f-ef1a-4a1f-9f5e",
 "actor": {
  "name": "Peter",
  "mbox": "mailto:some@new.user",
  "objectType": "Agent"
 },
 "verb": {
  "id": "http://.../verbs/solved",
  "display": {
   "und": "solved"
  }
 },
 "context": {
  "contextActivities": {
   "parent": [
    {
     "id": "http://../objects/problems",
     "objectType": "Activity"
    }
   ]
  }
 }}
```

More complex statement forms can be used and we will elaborate more on them in the next section. The set of verbs and objects an institution can work with is called *vocabulary*. Each institution can define its own vocabulary with no restriction as long as an URL links back each verb and object to a JSON stream describing it.

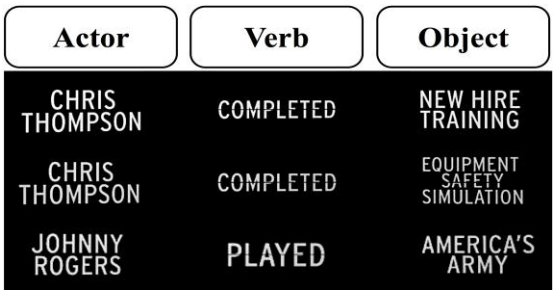


Fig. 4. Some examples of xAPI sentences

The Experience API was released, as version 1.0, in April 2013, and there are, as of today, over 100 adopters, projects and companies involved, such as those in Figure 5.

Fig. 5. Some adopters of the Experience API specification

The specification also contemplates a query API to help find logged statements, and performs some analytics (averages, aggregation, etc.) on the data. Finally, the Experience API is an open-source and free initiative, whose source code and specifications are open to anyone.

IV. EXPERIENCE API LRS AS AN ELEARNING MONITORING ENGINE

The core of the Experience API is the Learning Record Store (LRS). The LRS is a specific module for data storage that allows an LMS (or any other social platform) to report tracking information on the learning experience. At any time, an LMS can send collected data over the network to an Experience API web service. An LRS is nothing more and nothing less than a wrapper or API software layer to a SQL database (initially, a PostgresSQL instance in the original Rustici implementation), as can be appreciated from Figure 6. This free LRS implementation was open-sourced by ADL (available at its Github repository) and is based on the Python computer language and on the publicly acclaimed Django web framework.

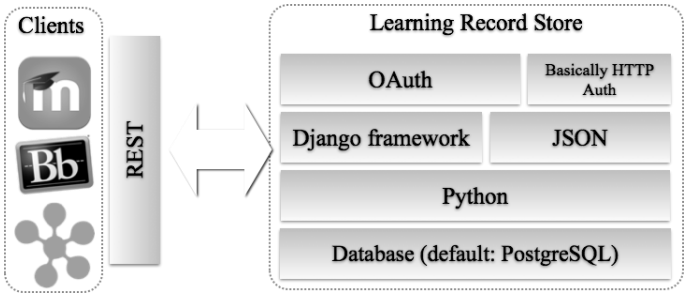


Fig. 6. Usual LRS software stack and interaction

The *learner (actor)*, *verb* and *object/activity* elements explained above are mandatory when talking to the LRS. However, they can be complemented with *result* and a *context* extra fields with additional information.

Students who interact with educational content via different systems or tools will leave traces in the LRS; each of these tools, if appropriately designed, will provide a totally different actor/user ID to preserve anonymity.

The *verb* element is a key part of an LRS communication, because it describes the action performed by the student. A URL must also be attached to the verb JSON property, pointing to its definition. This definition is composed of a name, a description, and a brief text suggesting plausible uses. In an eLearning environment, a verb is usually employed in its past tense form and could be something like: "read", "tried", "failed", "passed", "experienced", etc.

The *object/activity* part of the statement refers to "what" was experienced in the action defined in the verb, and usually corresponds to the learning activity (webinar, wiki, chat room, forum, mail message, etc.). Objects/activities must also embody a URL pointing to their rationale, which can include other information such as a description of the learning activity, verbs that can apply, possible results and usage suggestions.

The *result* component provides the denouement to the statement. It includes score, level of success and completion fields.

The context part adds more details to the overall statement, like the relationship of the activity with other activities, its order in the learning stream, or the teacher's name.

To every element in a sentence (actor, verb, context, etc.) sent to the LRS can be added, if needed, any type of pair key/value with extra information. It is even possible to add localization information so that an element can be perfectly identified in all possible languages.

As introduced in Figure 6, an LRS must also implement REST calls for data transfer (PUT, POST, GET and DELETE). The Experience API can make use of either OAuth or HTTP Basic Authentication when communicating with the outside world, ensuring a certified and secured dialogue between clients (usually an LMS) and the LRS service.

One of the key aspects of the LRS architecture is that it can be implemented in shared cloud ecosystems, allowing communications from very different eLearning platforms and academic institutions. In other words, monitoring data can be uniformly stored, allowing rapid, vast and democratic access to learning analytics information. Also, as LRS servers can integrate data from many different sources and from the same user/learner in a harmonized way, recommender systems can reduce the effects of possible cold-start scenarios.

Some companies are beginning to offer corporate cloud LRS services at different price tiers: Rustici Software, Saltbox, Learning Locker, Biscue, Clear, Grassblade, among others. Some also include compelling online analytics tools.

There exist some free LRS *hosting* services but mainly for testing and technology promotion purposes, and not applicable for research or production environments. It is worth mentioning the service run by ADL (lrs.adlnet.gov/xAPI) and the one deployed by Rustici Software (demo.tincanapi.com).

V. THE LIME MODEL AND THE LRS

Now that we have reviewed the most prominent monitoring techniques and introduced a few recent efforts towards regulation, we should ask how a real recommender engine could work with and benefit from a specific RDF-based source. The Experience API and the LIME model, explained below, are chosen.

The LIME model, presented in 38], is a tutor-lecturer-crafted rule-based recommender grounded on four separate pedagogical components strongly evident in all stages of education (Figure 7):

- Learning, or what every learner needs to do in order to assimilate and build knowledge on his or her own.
- Interaction, or relationships established, activities and academic interaction between students, leading to the acquisition of knowledge and competencies.
- Mentoring, or what teachers/tutors give relevance to.
- Evaluation, or officially graded activities, in every single category above listed.

Lecturers-tutors must design a strategy for each of his/her courses. The model codifies this strategy for a course or class group by using settings and categories.

A *course setting* is the balance between *formal* and *informal*

scenarios. In this context, *formal* means a regular academic programme with regular evaluation means (e.g. graded exams); *informal* means continuous evaluation and user activity inside the Learning Management System and every tool linked to it (e.g. Social Networks or repository). The system collects specific inputs from both settings, keeping an overall balance of 100%. For instance, if the designer requires just a *formal* setting, the balance should be *Informal*: 100% - *Formal*: 0%.

Furthermore, a learning scenario must be defined as the balance between the Learning, Interaction, Mentoring, and Evaluation, in combination with the Formal and Informal settings categories. In the LIME model, every category and setting are assigned with a specific weight (w_i), keeping an overall balance of 100%.

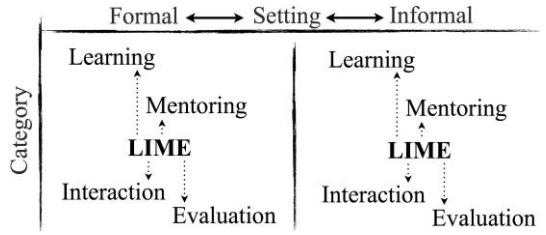
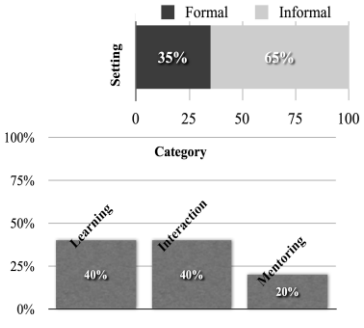


Fig. 7. Categories and settings in the LIME model

In the LIME model each input (action performed by a student in the eLearning platform or Social Network) is attributed a category and a weight, assigned by the teacher/tutor.

An example of model configuration for a specific site can be found in Figure 8. Based on these components, tutors can manually define and parameterize recommendation rules, which will only trigger a message to the student if conditions regarding categories, inputs and settings are met.



Inputs	Category (L-I-M)	Setting (In)Formal	EVAL
New content	L	I	
Forum post	I	F	
Problem set	L	F	✓
Direct message	I	I	
Progress test	M	F	
Chat message	I	I	
Profile update	I	F	
Public answer in forum	I	F	
Grade in test	M	F	✓

Fig. 8. Sample configuration of the LIME model for a specific course site

LIME is therefore a tutor-lecturer-crafted, rule-based recommender system for cloud-institutional learning environments (SPOCs or MOOCs), which contrasts with other recommendation paradigms reviewed in previous sections. LIME's goal is simply to improve learning efficiency, and to facilitate the learning itinerary of every student by a personalised recommendation set.

LIME can be fed from learner inputs in a variety of ways. However, our model can also be initialized with tracked data stored in a xAPI LRS instance/server if we make some assumptions.

How can LIME inputs be built out of information stored in the LRS? A LIME model input has to define an action and a context in which a learner performs this action:

- participation in chat
- answer in main forum thread
- message to tutor
- resolution of problem set
- formal broadcast mail to mates
- ratio of emoticons used in communications
- ...

xAPI verbs and objects, taken in an isolated way, are not sufficient. However, a joint entity composed of a verb plus an xAPI object makes more sense in our model, as shown in Figure 9:

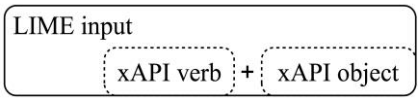


Fig. 9. LIME Inputs from xAPI sentences

As stated above, verbs and objects in the xAPI specification must be backed by JSON composites with information about meaning and usage tips. It is up to the implementer to define which verbs and objects best represent the scenario to be tracked and monitored. Let us take a look at the sample verbs and activities available on the official Experience API site (adlnet.gov/expapi). In Figure 10 are listed all the verbs and activities the LRS can store and their possible combinations to build a meaningful and compatible LIME input.

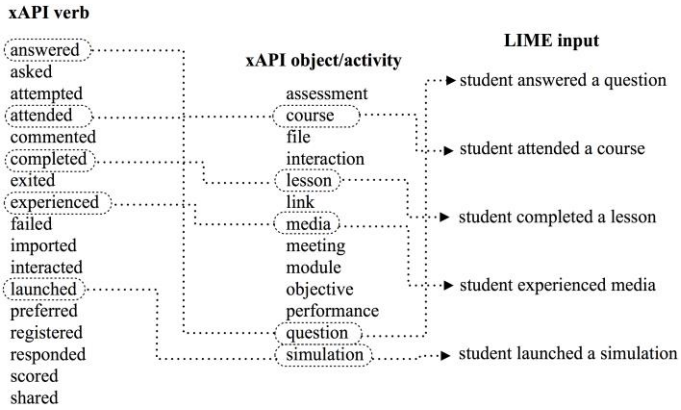


Fig. 10 From xAPI verbs and objects to LIME inputs

As explained in previous paragraphs and as part of the model configuration, each input should be assigned a weight (w_i), a category and a setting. These parameters should not reside on the LRS but on the LIME system's own configuration repository. In other words, LIME administrators should maintain an updated *equivalency list* between LRS vocabulary and LIME inputs. These inputs will then interplay with rules (Figure 11), which are, in turn, based on *predicate*

filtering. Predicates are applied over collections of inputs and highly resemble W3C XQuery or ECMA LINQ, detailed in [HYPERLINK \l "Saigaonkar:2010:XFS:1858378.1858429" 39],40] and [HYPERLINK \l "Pardede:jucs_15_10:sqlxml_hierarchical_" 41].

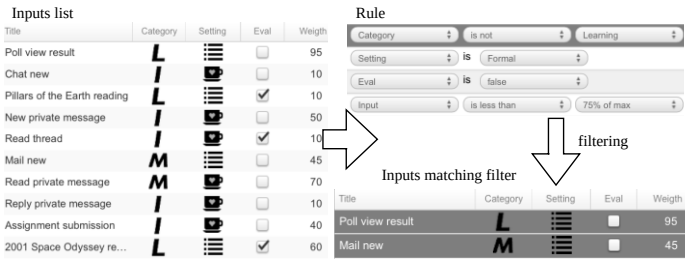


Fig. 11: Predicate filtering in LIME

As LIME was developed as a Basic Learning Tool Interoperability (Basic LTI) application, this equivalency list can even be stored in the LMS database through the LTI Settings API specification, part of LTI 1.0 and above. The model thus remains free from external configuration files or own database management. In order to save this list, it is only necessary to send a POST HTTP request like the one in the following example:

```
  POST http://server/imsblis/service/
  id=832823923899238
  lti_message_type=basic-lti-savesetting
  lti_version=LTI-1p0
  setting="participated+chat=message in chat
room; experienced+lesson=read text"
  oauth_callback=about:blank
  oauth_consumer_key=1213415
  oauth_nonce=14c6211cc66d87644f0855511
  oauth_signature=IkllkkZ1qfShYBYE+BhC
  oauth_signature_method=HMAC-SHA1
  oauth_timestamp=1338872426
  oauth_version=1.0
```

It is important to notice that LMS must be LTI compatible and support the Settings API protocol.

VI. LRS DATA AGGREGATION AND LIME RULES

Once LRS sentences are stored and an agreement between LIME inputs and these has been established, we have all the necessary ingredients to trigger recommender rules and deliver recommendations to students, if applicable. However, rules in LIME cannot operate upon atomic and individual LRS records, but only upon averages and aggregated substantial data, which offer a more equalized view of the learner situation. An example of this aggregation procedure is presented in Figure 12:

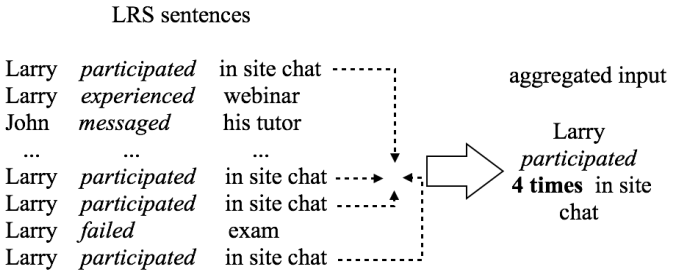


Figure 12: Aggregation of LRS sentences

Mathematically:

$$(\text{LIME input})_i = \frac{\sum_{j=0}^{n} (\text{LRS statement})_j}{w_i}$$

These aggregation operations are covered by the xAPI standard as well. The Experience API provides a query language to easily data-mine an LRS. For instance, the following code collects all the times the user "John" has tried an exam, and returns an aggregated result:

```
stmts.where(
    'actor.name = "John" and ('+
        'verb.id =
"http://adlnet.gov/expapi/verbs/passed"'+
        ' or '+
        'verb.id =
"http://adlnet.gov/expapi/verbs/failed"'+
    ')')
```

The default (and so far only) implementation of this query language is the ADL.Collection API, written in Javascript and ready to be used in browsers or on the server-side with NodeJS. There are two versions of this API: CollectionSync and CollectionAsync. They are almost the same, but the Async version runs the queries in a separate worker thread. The downside of this is that the statements must be serialized and passed into the worker, which can be slow. On the other hand, the user interface is more responsive.

VII. CONCLUSION

This paper describes incipient technologies and steps taken towards the dissemination of standardized monitoring engines. The engine mainly underlined in this paper is the Experience API, or xAPI for short. xAPI has been designed to store user data in a simple, centric, standard, client agnostic and powerful way. We also discuss the suitability of recommender systems in general and of the LIME recommender model in particular. LIME is a rule-based recommendation model. Rules in LIME require inputs (e.g. learner data and actions taken) that can be obtained in a variety of ways, like user tracking and interaction, user performance, or user profile.

We also perform a survey of the most common monitoring techniques and how they have been implemented in previous research projects related to recommender systems and learning analytics in general. With this review we illustrate there is no agreed way on how to register learner events. All mentioned

techniques incorporate a certain percentage of dependency on the system software being monitored.

Finally, we present the required adaptations and modifications that xAPI sentences need in order to build LIME-compatible inputs and how those can be aggregated and mined in order to feed system rules. On rule execution, our model delivers suggestions to students and learners. The xAPI spec atomizes learner actions in verbs and objects, which must be syntactically combined in order to obtain the aforementioned inputs. These combinations must be designed and listed by the tutor/teacher and handed over to our model. We suggest this equivalency list resides in the LMS's own database space, thanks to the LTI Settings API. The Experience API also offers native aggregation-statistical tools, which turn out to be of great help in this process.

ACKNOWLEDGMENT

This research is partially funded by UNIR Research (http://research.unir.net), Universidad Internacional de La Rioja (UNIR, http://www.unir.net), under the Research Support Strategy (2013-2015), Research Group TELSOCK.

VIII. REFERENCES

[1] G. Linden, B. Smith, and J. York, "Amazon.com recommendations: item-to-item collaborative filtering," Internet Computing, IEEE, vol. 7, no. 1, pp. 76-80, Jan 2003.

[2] F. Ricci, L. Rokach, B. Shapira, and P.B. Kantor, Recommender Systems Handbook.: Springer, 2010.

[3] Benjamin Marlin, "Modeling User Rating Profiles For Collaborative Filtering," in NIPS'03, 2003.

[4] J. Bobadilla, F. Ortega, A. Hernando, and A. Gutiérrez, "Recommender Systems Survey," Know.-Based Syst., vol. 46, pp. 109-132, Journal of Universal Computer Science 2013.

[5] M. Aberdour, "Open Source Learning Management Systems: Emerging open source LMS markets," 2007.

[6] Fran, Martin Ebner, Alexander Pohl, and Behnam Taraghi, "Interaction in Massive Courses," Journal of Universal Computer Science, vol. 20, no. 1, pp. 1-5, jan 2014.

[7] Y. Epelboin, "MOOC in Europe," UPMC-Sorbonne Université, 2013.

[8] Daniel Burgos, Colin Tattersall, and Rob Koper, "Representing Adaptive and Adaptable Units of Learning," in Computers and Education.: Springer Netherlands, 2007, pp. 41-56

[9] Jesus Bobadilla, Fernando Ortega, Antonio Hernando, and Javier Alcal, "Improving collaborative filtering recommender system results and performance using genetic algorithms," Knowledge-Based Systems, vol. 24, no. 8, pp. 1310-1316, 2011.

[10] M.AC. González, F.J.G. Penalvo, M.J.C. Guerrero, and M.A Forment, "Adapting LMS Architecture to the SOA: An Architectural Approach," in Internet and Web Applications and Services, 2009. ICIW '09. Fourth International Conference on, May 2009, pp. 322-327.

[11] S.K. Malik and S. A M Rizvi, "Information Extraction Using Web Usage Mining, Web Scrapping and Semantic Annotation," in Computational Intelligence and Communication Networks (CICN), 2011 International Conference on, Oct 2011, pp. 465-469.

[12] A Holmes and M. Kellogg, "Automating functional tests using Selenium," in Agile Conference, 2006, July 2006, pp. 6 pp.-275.

[13] Tomas Grigalis and Antanas , "Unsupervised Structured Data Extraction from Template-generated Web Pages," Journal of Universal Computer Science, vol. 20, no. 2, pp. 169-192, feb 2014

[14] H. Bosch et al., "Innovative filtering techniques and customized analytics tools," in Visual Analytics Science and Technology, 2009. VAST 2009. IEEE Symposium on, 2009

[15] P. Hunter, Instant Nokogiri.: Packt Publishing Ltd., 2013.

[16] Angel A. Juan, Thanasis Daradoumis, Javier Faulin, and Fatos Xhafa, "SAMOS a Model for Monitoring Studentsand Groups; Activities in Collaborative eLearning," Int. J. Learn. Technol., vol. 4, no. 1/2, pp. 53-72, 2009.

[17] Riccardo Mazza and Christian Milani, "GISMO: a Graphical Interactive Student Monitoring Tool for Course Management Systems," in T.E.L.'04 Technology Enhanced Learning '04 International Conference. Milan, 2004, pp. 18-19.

[18] Jungsoon Yoo, Sung Yoo, Chris Lance, and Judy Hankins, "Student Progress Monitoring Tool Using Treeview," SIGCSE Bull., vol. 38, no. 1, pp. 373-377, 2006.

[19] S. Shishehchi, S.Y. Banihashem, and N.AM. Zin, "A proposed semantic recommendation system for e-learning: A rule and ontology based e-learning recommendation system," in Information Technology (ITSim), 2010 International Symposium in, vol. 1, June 2010, pp. 1-5.

[20] K. Takano and Kin Fun Li, "An Adaptive e-Learning Recommender Based on User's Web-Browsing Behavior," in P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC), 2010 International Conference on, Nov 2010, pp. 123-131.

[21] K. Takano and Kin Fun Li, "An adaptive learning book system based on user's study interest," in Communications, Computers and Signal Processing (PacRim), 2011 IEEE Pacific Rim Conference on, Aug 2011, pp. 842-847.

[22] Enrique García, Crist, Sebasti, and Carlosde Castro, "An architecture for making recommendations to courseware authors using association rule mining and collaborative filtering," User Modeling and User-Adapted Interaction, vol. 19, no. 1-2, pp. 99-132, 2009.

[23] El Hassan, A. and El Adani, M. El Bachari E., "Design of an Adaptive E- Learning Model Based on Learner's Personality," Ubiquitous Computing and Communication Journal, vol. 5, 2010.

[24] C. Romero and S. Ventura, "Educational data mining: A survey from 1995 to 2005," Expert Systems with Applications, vol. 33, no. 1, pp. 135-146, 2007.

[25] Ahmad A. Kardan, Nahid Ghassabzadeh Saryazdi, and Hamed Mirashk, "Learner Clustering and Association Rule Mining for Content Recommendation in Self-Regulated Learning," International Journal of Computer Science Research and Application, 2012.

[26] Boban Vesin, Mirjana Ivanovi, Aleksandra Kla, and Zoran Budimac, "Protus 2.0: Ontology-based semantic recommendation in programming tutoring system," Expert Systems with Applications, vol. 39, no. 15, pp. 12229-12246, 2012.

[27] Tong Wang and Pi lian He, "Web Log Mining by an Improved AprioriAll Algorithm.," in WEC (2), 2005, pp. 97-100.

[28] M. K. Khribi M. Jemni, "Toward a Hybrid Recommender System for E-Learning Personalization Based on Web Usage Mining Techniques and Information Retrieval," in World Conference on E-Learning in Corporate, Government, Healthcare, and Higher Education, 2017.

[29] John C. Stamper et al., "Managing the Educational Dataset Lifecycle with DataShop," in Proceedings of the 15th International Conference on Artificial Intelligence in Education, Berlin, Heidelberg, 2011, pp. 557-559.

[30] Riccardo Mazza, Marco Bettoni, Marco Far, and Luca Mazzola, "MOCLog--Monitoring Online Courses with log data," Proceedings of the 1st Moodle Research Conference, pp. 14-15, 2012.

[31] JeffZ. Pan, "Resource Description Framework," International Handbooks on Information Systems, 2009.

[32] IMS Global Learning Consortium Inc., "Learning Measurement for Analytics Whitepaper," 2013.

[33] IEEE, "Data Model for Content to Learning Management System Communication," IEEE Std 1484.11.1-2004, 2005.

[34] J and Atkins, M and Norris, W and Messina, C and Wilkinson, M and Dolin, R Snell, "JSON Activity Streams 1.0," 2011.

[35] Abelardo Pardo and George Siemens, "Ethical and privacy principles for learning analytics," British Journal of Educational Technology, vol. 45, no. 3, 2014.

[36] David Kelly and Kevin Thorn, "Should Instructional Designers Care About the Tin Can API?," eLearn, vol. 2013, no. 3, 2013.

[37] A del Blanco, A Serrano, M. Freire, I Martinez-Ortiz, and B. Fernandez-Manjon, "E-Learning standards and learning analytics. Can data collection be improved by using standard data models?," in Global Engineering Education Conference (EDUCON), 2013 IEEE, March 2013, pp. 1255-1261.

[38] Daniel Burgos, "L.I.M.E. A recommendation model for informal and formal learning, engaged," IJIMAI, pp. 79-86, 2013.

[39] Swati Saigaonkar and Madhuri Rao, "XML Filtering System Based on Ontology," in Proceedings of the 1st Amrita ACM-W Celebration on Women in Computing in India, New York, NY, USA, 2010, pp. 51:1--51:6.

[40] James Cheney, Sam Lindley, and Philip Wadler, "A Practical Theory of Language-integrated Query," SIGPLAN Not., vol. 48, no. 9, pp. 403-416, 2013.

[41] Eric Pardede, J. Wenny Rahayu, Ramanpreet Kaur Aujla, and David Taniar, "SQL/XML Hierarchical Query Performance Analysis in an XML-Enabled Database System," Journal of Universal Computer Science, vol. 15, no. 10, pp. 2058-2077, may 2009.

M.Sc. Alberto Corbi works as a senior researcher at the Technology-enhanced Learning & Social Networks (TELSOCK) research group and as a teaching assistant at the School of Engineering, both part of the International University of La Rioja. With a background in physics (M.Sc. in ocean-atmosphere interaction), computing and education, he is currently involved in research fields around recommender systems, eLearning standards and systems interoperability. Simultaneously, he is working on his PhD thesis around medical imaging at the Spanish Council for Scientific Research (CSIC).

Prof. Dr. Daniel Burgos works as Vice-Chancellor for Research & Technology and UNESCO Chair on eLearning at the International University of La Rioja (www.unir.net, http://research.unir.net). Previously he worked as Director of Education Sector and Head of User Experience Lab in the Research & Innovation Department of Atos, Spain, and as an Assistant Professor at Open Universiteit Nederland before that.

His research interests are mainly focused on Adaptive and Informal eLearning, Learning & Social Networks, eGames, and eLearning Specifications. He is or has been involved in a number of European-funded R&D projects, such as Intuitel, Hotel, Edumotion, Inspiring Science Education Stellar, Gala, IntelLEO, Go-MyLife, Grapple, Unfold, ProLearn, TenCompetence, EU4ALL, NiHao, Kaleidoscope, Sister, and ComeIn. Prof. Dr. Burgos holds degrees in Communication (PhD), Computer Science (Dr. Ing), Education (PhD), and Business Administration. Furthermore, he is a member of a number of Executive Boards of associations and professional clusters focused on educational technology and eLearning innovation, for example Telearc, Telspain, Menon Network, Adie, and others.