*Journal of Universal Computer Science, vol. 19, no. 14 (2013), 2075-2092
submitted: 3/2/13, accepted: 19/7/13, appeared: 1/8/13*

A Generic Architecture for Emotion-based Recommender Systems in Cloud Learning Environments

Derick Leony
(Universidad Carlos III de Madrid, Spain
dleony@it.uc3m.es)

Hugo A. Parada Gélvez
(Universidad Carlos III de Madrid, Spain
hparada@it.uc3m.es)

Pedro J. Muñoz-Merino
(Universidad Carlos III de Madrid, Spain
pedmume@it.uc3m.es)

Abelardo Pardo
(The University of Sydney, Australia
abelardo.pardo@sydney.edu.au)

Carlos Delgado Kloos
(Universidad Carlos III de Madrid, Spain
cdk@it.uc3m.es)

Abstract: Cloud technology has provided a set of tools to learners and tutors to create a virtual personal learning environment. As these tools only support basic tasks, users of learning environments are looking for specialized tools to exploit the uncountable learning elements available on the internet. Thus, one of the most common functionalities in cloud-based learning environments is the recommendation of learning elements and several approaches have been proposed to deploy recommender systems into an educational environment. Currently, there is an increasing interest in including affective information into the process to generate the recommendations for the learner; and services offering this functionality on cloud environments are scarce. Hence in this paper, we propose a generic cloud-based architecture for a system that recommends learning elements according to the affective state of the learner. Furthermore, we provide the description of some use cases along with the details of the implementation of one of them. We also provide a discussion on the advantages and disadvantages of the proposal.
Key Words: recommender systems, affective computing, cloud learning environments, software architecture
Category: D.2.11, H.3.3, L.3.2

1 Introduction

Online learning environments have evolved from tutors and learners using isolated tools on internet, to become into a set of available tools on internet allowing ubiquitous access; this is called Cloud Learning Environments (CLE). The CLE have allowed learners to create a virtual personal learning environment (PLE). This PLE are mainly based on free and generic tools that support basic tasks. An important characteristic of learning environments in the cloud is the personalization of the environment according to the learner needs, objectives and current situation. Thus, the learning environment can adapt its interface, content, and capabilities according to personal characteristics of the learner: current learning objectives, learning achievements, skills, preferences and affective state. In addition, contextual information is another input for personalization: current location, time of the day and available technology.

Recommender systems are among the instruments used to provide learning environments with personalization.This occurs by suggesting learning resources that are most suitable for the characteristics and behaviours of the learner. For example, a learner that has accessed material to learn the basics of Spanish may be interested in other resources related to the subject. Later, she may benefit from being offered with material of intermediate Spanish. Manouselis et al. [Manouselis et al., 2012] describe the approaches followed so far to implement and deploy recommender systems in Technology Enhanced Learning (TEL).

More recently, some approaches are including affective information of the learner, such as the case of the Semantic Affective Educational Recommender System proposed by Santos and Boticario [Santos and Boticario, 2012]. The affective information is captured by several means, most of them imply using body sensors to measure physiological characteristics like the electrical activity of the heart or the brain. Other approaches involve the analysis of facial gestures in order to detect affective expressions. In [Calvo and D'Mello, 2010] the authors provide an exhaustive review of the state of the art with further information of models and methods to detect affective states.

In this paper, we present an improvement of the architecture presented in [Leony et al., 2012] of the Learning Resource Affective Recommender (LRAR). The most relevant change for the cloud context is the deployment of the recommending engine as an auto-scaling service, which allows to serve several clients with reasonable response time and performance. Thus, the recommender engine is transformed into a recommending service based on cloud technologies. The migration to the cloud allows us to improve the scalability of the recommender system, given the high level of computational power required by this kind of processing.

The rest of the paper is structured as follows: section 2 presents the main works related with our proposal. In section 3 describes the proposal of a cloud-based generic architecture for a learning recommender system aware of or learners' affective states. Section 4 provides the details of three use cases for the architecture: recommendation of users, recommendation of learning activities and recommendation of learning resources. Finally, section 5 presents a discussion about advantages and weaknesses of the architecture while section 6 describes our future work.

2 Related works

The adoption of cloud computing has risen constantly during the last years [Keswani et al., 2012]. This trend also is observed in the educational arena where research on cloud in learning technologies has increased in last few years. This increase of interest on cloud in learning technologies has enabled a shift from monolithic systems into flexible e-learning services [Dagger et al., 2007] and the proliferation of service-oriented e-learning environments [Muñoz Organero et al., 2010] where e.g. personalized educational services are given in a distributed way but not in a unique central Learning Management System. In this section we present some of the most recent works regarding the application of cloud technology into the learning domain.

Cloud technologies are a set of easily accessible and virtualized resources that can be dynamically adapted allowing an optimum resource utilization [Vaquero et al., 2009]. A cloud technology allows users access onto different services on internet through diverse devices. It also provides service providers with several advantages, such as availability, integration of multiple services, flexibility and scalability [Leavitt, 2009]. This makes cloud technology an attractive option for deploying applications that require a high level of computational power. Hence cloud technology has taken into account to support services that offer educational resources to academic communities.

In this sense, in [Alexander, 2012] Mikroyannidis states that the cloud offers a lot of services for building adaptive and customisable CLE. He also explains how CLEs extend the borders of the learning environments beyond of educational organization. Additionally this work proposes a learning scenario based on the use of cloud learning services. Thus, to take advantage of these features educational institutions are also moving towards providing their services by using cloud technologies. However, in the educational domain, services are scarcely adapted and offered in cloud.

Some distributed oriented learning architectures have been proposed such as OKI [OKI, 2012], KnowledgeTree [Brusilovsky, 2004], or in the ELENA project [Dolog et al., 2003]. OKI (Open Knowledge Initiative) proposes an architecture based on layers that includes educational services that can be accessible invoking remotely a set of web services. OKI defines and provides the interfaces for these services, and can create final applications that offer a user interface, combining different invocations from these services. OKI is focused on educational services that can be found in a typical LMS such as assessment or authorization, and the invocation of the implementation of these interfaces can lead to build a specific LMS. However, OKI is not focused on typical adaptive e-learning services. On the other hand, KnowledgeTree [Brusilovsky, 2004] and ELENA [Dolog et al., 2003] both represent distributed architectures for the personalization of resources for technology enhanced learning. Nevertheless, these architectures focus on generic aspects of adaptation of e-learning resources, but do not go into details of the specifics for affective learning. KnowledgeTree and ELENA would be complementary to our work as they can be used as general frameworks for distributed adaptive learning, while our architecture would bring the details for affective learning. In addition, OKI, KnowledgeTree and ELENA do not focus on analyzing important aspects of cloud computing as the distributed computation of a task, which is also addressed in this work.

Several approaches of cloud architectures promote improvements to services in the e-learning area by using cloud technologies. Thus, they try to overcome challenges faced by educational institutions. In [Huang, 2012] Masud and Huang propose an e-learning cloud architecture to allow the migration of e-learning systems from schools to a cloud computing infrastructure. They describe an e-learning cloud architecture made up of five layers: infrastructure, software resource, resource management, service and application. This proposal describes a general architecture for e-learning; nevertheless it does not focus on how to implement an e-learning service in a cloud architecture.

As CLE appears as a set of available tools on the internet that allows ubiquitous access to an academic community, it is evident that the existing of Personal Learning Environments in the cloud are in an early stage of its developing. Currently the CLE has dealt in offering an environment that allows learners and teachers easy access to different tools for producing and consuming academic content. A related work is also presented by Al-Zoube in [Al-zoube, 2009], where he proposes a cloud computing based solution for building a virtual PLE. This proposal consists of allowing the learner access to different tools offered on internet such as iGoogle, Google docs, YouTube, etc. however such as Stein et al. state in [Stein et al., 2012] public clouds generally meet the common base of user requests, but they may not be designed to meet educational needs. In addition, today learners are demanding specialized learning services in order to improve the learning results. Hence, the educational domain requires design and deploy services in order to built a true educational Cloud.

Following the above approach, in [Deepanshu Madan, 2012] Madan et al. present a cloud-based learning service model. This proposal describes comprehensively the cloud computing services as a key aspect of cloud computing model. The authors focus on services and available models to be deployed into cloud architecture. They claim that institutions should use the existing cloud infrastructure offered by companies such Google, Amazon and others. Then educational institutions should focus on defining the cloud service layer to implement it into the cloud architecture.

There is a known necessity of building a CLE based on specialized services rather than the traditional tools found in PLEs. However as the necessities of resources and services for learners and teachers are variable, we must offer a service to adapt the PLE in the cloud according to these necessities. In other words, CLE needs a recommender system to fill this gap. Recommender systems have been extensively deployed, however few systems operate in the education arena [Verbert et al., 2012]. Then in this work we propose a generic architecture for a system in the cloud to recommend learning elements according to the affective state of the learner. Thus learners will be able to adapt their PLE and CLE. Additionally we provide details of the architecture implementation of this specialized service.

3 Description of the generic architecture

The purpose of the proposed architecture is to deliver a set of recommended learning meta-data following a Software as a Service approach and based on affective information of the learner. The meta-data provides information about any element involved in a learning scenario: resources, activities, learners, instructors or feedback; we refer to all of these as *learning elements*. The architecture is composed by two layers: a service layer that executes the storage and recommendation tasks, and a client layer embedded in a learning environment. The details of each layer and their communication are described as follows.

3.1 Service layer

The service layer is in charge of receiving petitions from several clients and doing the requested tasks. The available tasks are to update the affective information of a learner, to update the information of a learning element, and to recommend learning element for a given learner. The first two tasks represent an administration interface for the management of learners' affective states and learning elements. Recommending learning elements is the main task of the service and it is the one that consumes most computational resources.

The service layer includes two storage elements to keep the information of the learning elements and the learners' affective states. The storage of learning elements is divided in categorical data-sources for each of the possible types of elements to store: users (learners or instructors, learning resources with content, learning activities, and feedback. These data sources only include meta-data and not actual content. This decision relies on the fact that the recommendation service is not meant to act as a repository of learning elements but just as a referrer.

The format of the database can be any usual specification for each specific type of element. For example, learning resources can make use of Learning Object Meta-data (LOM) or SCORM. The description of learning activities can be done with specifications like IMS Learning Design or Simple Sequencing. Plus, relationships between learners and instructor can be indicated through Friend-of-a-Friend (FOAF). The specifications of the format to use depend on the use case and are left to be decided by the implementation of the architecture.

The storage for learners' affective state is updated by requests from the client layer. The service is ready to create a new learner profile which consists of the learner identifier, current affective state and the record of the interactions with the elements to recommend and the affective state presented when that interaction occurred. The format used to define the affective state is decided by the implementation of the architecture as well. The specification EmotionML should be strongly considered because, although still being a W3C Candidate Recommendation, it allows flexible and complete definitions of affective states.

Besides the storage elements, the service layer has a recommendation engine cluster. The engine is designed as a cluster because the task to generate recommendations is the most expensive in terms of computational resources, which makes it the critical process to scale. The cluster contains as many instances of a recommending engine node as needed to support the service demand at the moment. Each node is in charge of analyzing the recommended elements, the affective states of a learner and generate assign a relevance score to each element the learner has not interacted yet. The algorithm used to calculate the recommendations is also left for the implementation of a specific use case.

3.2 Client layer

In the presented architecture, the client layer is represented by the learning environment and the instructor environment, since both can make use of the recommendation service to provide adaptation. The inclusion of recommendations is done by an embedded element deployed within each environment. The embedded elements communicate with the service layer to send and request information related to the recommendation based on affective states.

The element embedded in the learning environment sends the learner identifier and her affective state to the service layer. The client layer also informs to the service when a learner accesses a learning element. Optionally, the client can also be in charge of detecting the learning state of the learner and updates the affective state database.

There can be two moments when the embedded element requests information from the service layer. First, when accessing the learning environment the first time the client requests the last known affective state and the last recommended elements. The second moment is right after the affective status of the learner is updated. This is because a modification of the affective status implies a new set of recommendations for the learner, so the embedded element requests the list of recommended elements. After fetched, the list is displayed showing the title and description of each learning element. Then, the learner is able to select a resource based on its description or on the relevance score given by the recommendation engine. Figure 1, shows a diagram of the proposed architecture. The layers are displayed from bottom to top.

Being a generic architecture, it must consider a situation where elements are recommended to the instructor. This would be a case where the instructor receives information about learners that need an intervention, or feedback to provide to a specific learner. In this situation, the service layer connects not to a module embedded in the learning environment but in an environment that supports the instructor process. The functionalities of this embedded element are the same than the ones embedded in the learner environment, with the only difference of the user being the instructor.

The communication between the service and client layers is performed through the implementation of two Application Programming Interfaces (APIS): the Generic Recommender API and the Affective API whose purposes and methods are explained in detail as follows.

3.3 Generic Recommender API

The Generic Recommender Application Programming Interface (GRAPI) allows to manage the elements to recommend, and the interactions of the learners with the recommendations. Each of these management functionalities is explained as follows.

1. Management of element. This part of the GRAPI is in charge of the creation of new elements to recommend as well of updating their information. The methods considered for this part are:

   CreateElement Creates a new element given its type (resource, activity or feedback), identifier, title, description and URL for fetching content if any, along with any other information.

   UpdateElement Updates the information of an existing element.

   RemoveElement This method deletes a element from the system.

   GetRecommendedElement It gets all the elements in the system that are recommended for a given learner.

2. Management learner interactions. This part of the GRAPI is in charge of the tracking the interactions between learners and the elements that they are recommended. The methods defined in this part are as follows:

Figure 1: Generic architecture of a affective learning recommender system based on the cloud.

ViewElementStart Records the event of a learner starting viewing a recommended element.

ViewElementStop Indicates the action of a learner stopping the viewing of a recommended element.

MarkElement This method is used to indicate that the learner has marked a recommended element as favorite or relevant..

ViewElementUrl Indicates that a learner has accessed the URL of a recommended element.

3.4 Affective API

The Affective Application Programming Interface (AAPI) allows the management of emotions, the present state of emotions for each user, and the evolution of emotions for each user during all the different actions in his learning process.

The AAPI can be divided into 4 different parts depending on their function:

1. Management of emotions. It is in charge of defining the different considered emotions. An emotion should be some aspect of the affective state of a person. Every emotion should have a value in the interval [0, 1]. For example, some types of emotions can be frustration or happiness. A value of 0 in happiness would mean that the user is very sad, while a value of 1 indicates that this user cannot be happier. The methods of the AAPI for this category are:

   CreateNewEmotion Creates a new type of emotion given by its name, description, and other possible information related to that emotion.

   UpdateExistingEmotion Updates the information of an existing emotion.

   RemoveExistingEmotion This method removes an existing emotion in the framework.

   GetExistingEmotions Gets all the defined emotions in the environment.

2. Management of learners. Defines the different learners with their information and related emotions to be considered. The methods of the AAPI for this category are:

   CreateNewLearner Creates a new learner with all her related information (name, surname, description, etc.). Other information such as the one related to IEEE-PAPI or IMS-LIP can be aggregated but would be part of another API but not of the affective one.

   UpdateExistingLearner Updates the information of an existing learner.

   RemoveExistingLearner Removes an existing learner.

   GetExistingLearners Get all the learners of the framework.

   AssignEmotionsToLearners With this method, it is possible to define or update the considered emotions that will be retrieved for a learner. For example, we could consider for some educative experience to retrieve happiness, frustration, and surprise, but for other context only consider one of them.

3. Updating of learners' emotions. This category enables to set the values of the emotions for each one of the learners during the time. For each learner, the affective storage will store learners current emotions and learners emotions when they accessed some learning resource or activity, received some feedback or interacted with another user for requesting help or giving assistance. Therefore, there will be stored as many emotional states as the sum of contents, activities, requests for help and assistance for help that the learner interacted. The methods of the AAPI for this category are:

   SetInitialEmotion This method will set all the values for all assigned emotions for a learner initially, i.e. before making any interaction in the learning system.

   SetEmotionToLearnerInteraction This method will set all the values for all assigned emotions for a learner just after that learner finished the interaction with something or somebody in the learning process. The related information of the interaction should be stored and this should include the type of interaction (content, activity, request for help, assistance for others, feedback), the specific element which with they interacted, the time it took place, etc.

4. Retrieving of learners' emotions. This category allows to gather information about all the emotions of a learner while interacting in the learning environment. The methods of the A learner API for this category are:

   GetEmotionalState This method allows to retrieve the values of the assigned emotions for a specific learner after his interaction with a specific content, activity, giving assistance, receiving feedback or requesting for help.

   GetAllEmotions Provides all the assigned emotions of a learner during their learning process, giving the associated information of the learning content, activity, etc. related to each given emotion.

4 Use cases

In this section we describe three possible use cases that are supported by the generic architecture previously described. These are the recommendation of users, recommendation of activities, and the recommendation of learning resources which has been implemented as a proof of concept.

4.1 Recommendation of users

When a learner is blocked with some content or activity, e.g. he does not solve correctly an exercise or she does not understand well a specific theoretical content, this learner can request for help to other learners or even some teacher. In this case, it is a challenge to select the best person that can help and guide that learner. The issue is to recommend the most suitable user that can help the learner with problems.

Although typical information about the learning process is relevant for this recommendation such as the concepts and topics that a learner masters, however affective information about users is very relevant, as we might want to choose e.g. someone that is in a good mood in that moment so that she can offer the best help for the learner.

In our use case, when a learner requests for a recommendation of another user that can help himself in their learning process, the recommendation algorithm will take into account the following different aspects:

1. Measure of the convenience for being a user that gives assistance taking into account the present affective state. The idea is to find a user whose present affective state is good for helping another user. The different present values of the defined emotions will be taking into account with a weight according to their importance for this issue.

2. Measure of the level of knowledge of the user in the concepts of the resource. The idea is to find a user whose level of knowledge is adequate for the covered concepts in the element (content, activity, etc.) where the initial user was blocked. The knowledge level for each concept can also have a weight depending on its importance.

3. Measure of the similarity of the emotion for the same element. The idea is to select a user that has already passed for the same element where the learner is blocked and whose affective state in that moment was similar to the learner that is blocked. In that way, a learner that felt the same sensations can offer better help to this learner.

All of these 3 aspects will give a measure of adequateness and finally a global parameter will be given as a combination of these three aspects in a weighted way depending on their importance. In addition, if the final level of this parameter is not greater than a threshold, then a learner will not be recommended for giving assistance, but a teacher. This is because if it is considered that any learner did not have the minimum required level to provide help in that moment, then a teacher should act and intervene.

4.2 Recommendation of learning activities

The affective state of a learner is also a good indicator of the type of learning activity that the learner could perform. If the learner is bored, the recommender system could recommend activities that provoke an interaction such as solving a problem or discussing an issue with a group of classmates. However, if the learner is feeling confused, she could be recommended to read explanatory material related to the topics she is studying.

Learning activities can usually be associated to an affective state that the learner must present initially in order to carry it out. Thus, a rule-based algorithm is needed in order to recommend an activity that best fits the current situation of the learner. The aspects that take part of the condition to apply a rule will be the affective state, the learning profile (learning objectives, knowledge levels, etc.) and the history of activities already performed in order to satisfy possible dependencies. Once the learner completes the activity, her affective state and learning outcomes will be recorded. This will allow a validation of the recommendations and their effect on learning gains. In addition, this information will be valuable for future recommendations to similar learners.

4.3 Recommendation of learning resources

In this use case, the recommendation process within a recommender node follows the method known as user-based collaborative filtering. When a recommendation has to be done for a given learner, it first finds a set of the learner's *neighbors*, learners with similar patterns of access to resources. The level of similarity is to identify the neighbors of a learner is defined by a similarity function. In the case of the affective recommendation, the similarity of two learners is proportional to the amount of resources accessed by the learners when indicating the same affective state.

As the recommendations must take into account the affective state of the learner, the collaborative filtering process had to be extended to include that contextual information. The set of resources available for recommendation are a combination of the learning resources with the affective states. Thus,

$$R = L \times A$$

where R is the set of recommendable resources, L is the complete set of resources meta-data and A is the complete set of affective states. The recommendable resources are a tuple of a resource and an affective state.

After the learner's neighborhood is defined, learning resources are sorted based on how relevant they have been within that neighborhood of users. The resulting list must be filtered because it contains recommended tuples *(resource, affective state)* and the affective state might be any stored one. Thus, the list of recommended resources are only those that appear in a tuple where the affective state is the same one the learner presents at the moment.

The environment where we have implemented the affective recommender service is Amazon Elastic Computing Cloud (EC2), which is part of Amazon Web

Services (AWS). EC2 allows us to define an image that acts as a blueprint to generate several instances of a computer with the same software configuration. Each one of these instances is what in the definition of the architecture we have called a node.

In our implementation, the storage element for learner resources meta-data is implemented as a database deployed in the engine MySQL. The same database implements the learner affective state storage element. Since the data might be accessed from many nodes of the cluster, the database engine is installed in an independent node, not meant to be part of the recommendation cluster. In order to manage the information stored in the database, the database node implements a RESTful web services, while the technology supporting the web application is J2EE.

The recommendation engine is developed on top of Apache Mahout machine learning engine. Mahout provides a set of libraries to implement machine learning models such as collaborative filtering, recommender systems, clustering, pattern mining and classifiers. Mahout is implemented in Java and this allows a straightforward integration with a web application developed with J2EE. As explained in [Owen et al., 2011], Mahout is conceived to be scalable through the framework for distributed processing Apache Hadoop, which allows the definition of clusters of computers with computational and storage capabilities.

The algorithms for collaborative filtering use the map/reduce paradigm. This paradigm consists in two processes that can be parallelized and distributed among several computers to increase their speed. The *map* process generates a sequence of pairs where usually the first element is an entity identifier and the second element is an entity characteristic that will be needed in a further computation. The *reduce* process receives the pairs generated by the map process and computes an incremental value associated with the entity. For example, the first step to identify the neighbors of a learner is to identify the most frequent occurrences of a pair *(affective state, learner resource)* for each user. In this case, the map process returns a pair with the syntax *(user identifier, (affective state, learner resource)).* Thus, the entity to identify is the user and the other item to include is the pair or affective state an resource. The reduce process receives the same pair and create an array for each received user. The elements of the array are complex structures with the syntax *((affective state, learner resource), count)*, so that by sorting the array in descendant order by the second element we obtain the top occurrences of pairs for the given learner.

The advantage of using the map/reduce paradigm is that both process can be done in parallel and in several computers. This allows the implementation to scale by just creating a new node with the same characteristics of a previous recommender node. Hadoop keeps control of the nodes that are available in the cluster to perform computational and storage tasks. Thus, we are provided with a simple way to auto-adjusting the size of the recommender cluster by just adding or removing nodes according to the service demand.

For the implementation of the client layer element, a widget has been developed as a proof-of-concept of a tool that interacts with the affective recommender service. The widget has been developed using the Software Development Kit (SDK) provided by ROLE Project [Consortium, 2009-2012]. ROLE aims to provide the learner with a framework to build her Personalized Learning Environment. The widget is implemented in JavaScript and HTML, and it follows the OpenSocial Gadget specification.

The widget interface has two functional sections represented by the tabs. Resources, the main tab, allows the learner to state her affective state from a static list provided by the recommendation service. Currently, the list is based on the affective states used by D'Mello et al. in [D'Mello et al., 2007]; these include frustrated, confused, bored, enthusiastic, motivated and the normal state, meaning that there is no relevant affective state at the moment.

Resources tab also presents a list of learning resources ordered by relevance for the learner in her current state. Thus, once the learner submits a change on her affective state, the widget send a recommendation request to the affective recommender service. When the response is received, the client analyzes the list of resources and embed their information as the list of recommended resources.

Second section is the *Profile* tab, where a time-line of the affective states reported by the learner is embedded. Its objective is to provide the learner with a visualization of her emotional changes during the learning activity being performed. The log of affective states is also provided by the learning resource service.

Finally, the Settings tab allows the learner to set her learning objectives. These might be changed during the learning activity, which also triggers a change of the learning resources that are recommended. Figure 2 presents a screen capture of the widget deployed in ROLE environment, with emphasis on the resources recommended to a frustrated learner.

A scenario that exemplifies the use of the widget is the following. Alice, a university learner whose major is Computer Science, is trying to complete a C programming task that she was assigned as homework. She starts working highly motivated on the initial details of the program and she reflects be selecting the *Motivated* option among the affective states available in the widget. The widget communicates the affective state to the recommender service and this returns a list of resources suitable for Alice, such as C programming references and the user's manual of the compilation tool. Alice finds the resources helpful and uses them to write her code more quickly. As Alice advances she realizes that the task is not as easy as she first thought and that she might even encounter some programming errors that she was not expecting, this causes her affective state

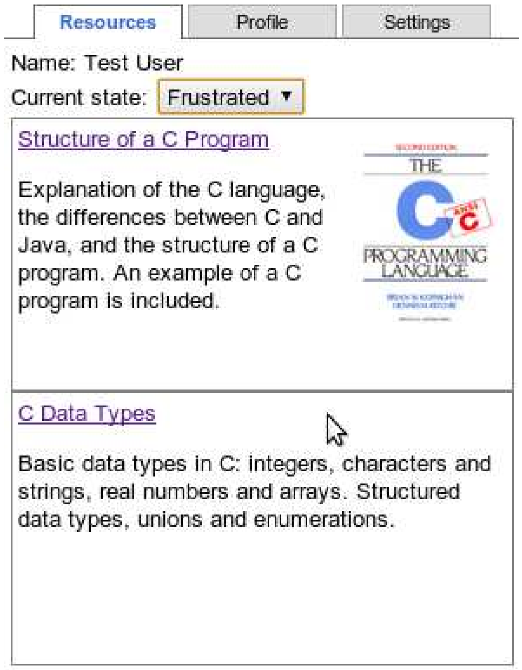

Figure 2: Implementation of the resource visualizer in the ROLE PLE.

to change. When she find that she cannot fix a compilation error she starts feeling frustrated. Thus, she updates her affective state to *Frustrated*. Again, the widget communicates the new affective state to the recommender service, obtains the list of suitable resources and displays them as part of the widget content. The new list of recommended resources includes basic programming concepts and common programming errors with their respective solutions (see Figure 2). Alice accesses the resources and solves the error of her program. After seeing that the widget helps her along the task, Alice recovers her positive mood and continues her work to finish the homework.

5 Discussion

An advantage of the proposed architecture is the flexibility provided in two levels of the implementation. The first level corresponds to the possibility to recommend to the learner many kinds of elements like learner resources, activities or peers. The second level of flexibility is related to the algorithms used to identify the relevant elements. The generality of the architecture is exemplified with varied concrete use cases.

Another advantage of the architecture is the high scalability to support large amounts of learners and elements to recommend. This scalability is supported by the use of cloud computing technologies such as AWS EC2 to adapt the computational resources depending on the load that the system is experiencing. However and as it is described further below, there is a need to evaluate the scalability with stress tests.

Furthermore, there are some issues that must be taken into account as possible disadvantages of the architecture such as approaches to caching and privacy issues. The first issue is the difficulty to cache results in the presented architecture. Given that in a cloud recommendation service the computational workload is distributed among the nodes of the cluster, one node cannot cache the recommendation calculate in another node. This issue can be addressed either at the client layer, where the client itself would be in charge of caching the information of the recommended elements. The cache techniques to use in order to select the most important recommendations in this context is a relevant topic.

Another possible approach to solve this issue is through the inclusion of a middle layer that caches and dispatches the recommendations; this layer would be between the recommendation cluster and the interface of the service. Another issue are the formats to use (XML, RDF, etc.) in the framework for storing and exchanging the information. The proposed architecture is generic and admits different formats of the information. The selection of one format or another might depend on the formats that admits the tools that implement the different services of the API, or the defined level of interoperability with other external entities.

Another concern is related to privacy issues, since the affective state of the learner is stored in a database accessible from several recommender nodes. The key in this issue is that the recommending nodes are the only elements with permission to access the affective state information. Furthermore, the recommender service does not require to store information that identifies the learner immediately, such as the full name or email. Instead, the service can use a hashed identifier of the learner and that would not interfere with the process of recommending learning elements.

6 Future work

Future work consists on evaluating the performance of the implementation presented in the article. The evaluation objective is to analyze the improvement on the response time of a recommendation request to the server. Part of this work includes to analyze the correlation between the service performance and the amount of recommender nodes in the cluster; this would lead to a set of guidelines for deploying the recommender service in a real learning scenario. In order to address the response time problem, a solution might be to process in advance in the background future recommendations. A challenge here is how to determine the states that are more probable in the future, so that the recommendations can be available in advance, avoiding the response time problem. This issue is similar to the one discussed in the architecture in [Muñoz Merino and Delgado Kloos, 2008].

Another line of work consists in the development of plug-ins to include sensors as a method to populate the affective state database. Specifically we are working with sensors for galvanic skin response and the recognition of face gestures through a video camera. These sensors have been proven to detect affective states with accuracy [Picard, 2000] and thus might be improve the recommendation process. They would also allow the learner to focus on the retrieval and use of elements rather than constantly informing her current affective state. On the same track, it is also intended to improve the interface for the learner to provide her affective state. Several approaches will be taken in order to obtain contextual information about what provoked a given emotion in the learner and how did the recommendation of elements interferes her affective state.

Acknowledgment

Work partially funded by the EEE project, "Plan Nacional de I+D+I TIN2011-28308-C03-01", the "Emadrid: Investigación y desarrollo de tecnologías para el e-learning en la Comunidad de Madrid" project (S2009/TIC-1650), and "Consejo Social - Universidad Carlos III de Madrid".

References

[Al-zoube, 2009] Al-zoube, M.: "E-Learning on the Cloud". International Arab Journal of e-Technology, 1, 2 (2009):58–64.

[Alexander, 2012] Alexander, M.: "A Semantic framework for cloud learning environments". In: Chao, Lee ed. Cloud Computing for Teaching and Learning: Strategies and Implementation. Hershey, PA: IGI Global (2012):17–31.

[Brusilovsky, 2004] Brusilovsky, P.: "Knowledgetree: A distributed architecture for adaptivee-learning". In Proceedings of the 13t international World Wide Web conference (2004), 104–111.

[Calvo and D'Mello, 2010] Calvo, R. A., D'Mello, S.: "Affect detection: An interdisciplinary review of models, methods, and their applications". Affective Computing, IEEE Transactions on, 1, 1 (2010):18–37.

[Consortium, 2009-2012] Consortium, R.: "ROLE Project". http://www.role-project.eu/ (2009-2012). Last visited September 2012.

[Dagger et al., 2007] Dagger, D., O'Connor, A., Lawless, S., Walsh, E., P., W. V.: "Service-oriented elearning platforms: From monolithic systems to flexible services". IEEE Internet Computing, 11, 3 (2007):28–35.

[Deepanshu Madan, 2012] Deepanshu Madan, A. A., Suneet Kumar: "E-learning based on Cloud Computing". International Journal of Advanced Research in Computer Science and Software Engineering (IJARCSSE), 2, 2 (2012).

[D'Mello et al., 2007] D'Mello, S., Graesser, A., Picard, R.: "Toward an affect-sensitive autotutor". Intelligent Systems, IEEE, 22, 4 (2007):53–61.

[Dolog et al., 2003] Dolog, P., Gavriloaie, R., Nejdl, W., Brase, J.: "Integrating adaptive hypermedia techniques and open rdf-based environments". In Proc. of The Twelfth International World Wide Web Conference, WWW 2003 (2003), 88–98.

[Huang, 2012] Huang, X.: "An E-learning System Architecture based on Cloud Computing". International Journal of Advanced Research in Computer Science and Software Engineering (IJARCSSE), 2 (2012):74–78.

[Keswani et al., 2012] Keswani, S., Krans, A., Henlin, E. H., Mirandi, J., Casey, M., Gagnon, K.: "4Q11 Public Cloud Market Landscape". Coud business quarterly, Technology Business Research, Inc. (2012).

[Leavitt, 2009] Leavitt, N.: "Is cloud computing really ready for prime time?" Computer, 42, 1 (2009):15–20.

[Leony et al., 2012] Leony, D., Pardo, A., Parada G., H. A., Delgado Kloos, C.: "A widget to recommend learning resources based on the learner affective state". In Proceedings of the 2nd and 3rd International Workshops on Motivational and Affective Aspects (2012).

[Manouselis et al., 2012] Manouselis, N., Drachsler, H., Verbert, K., Duval, E.: "Recommender systems for learning". SpringerBriefs in Computer Science (2012).

[Muñoz Merino and Delgado Kloos, 2008] Muñoz Merino, P., Delgado Kloos, C.: "An architecture for combining semantic web techniques with intelligent tutoring systems". In Proceedings of the Intelligent Tutoring Systems conference (B. P. Woolf, E. Aïmeur, R. Nkambou, and S. P. Lajoie, eds.), vol. 5091 of Lecture Notes in Computer Science. Springer (2008), 540–550.

[Muñoz Organero et al., 2010] Muñoz Organero, M., Munoz-Merino, P. J., Delgado Kloos, C.: "Personalized service-oriented e-learning environments". IEEE Internet Computing, 14, 2 (2010):i62–67.

[OKI, 2012] OKI: "OKI (open knowledge initiative)" (2012). URL `http://sourceforge.net/projects/okiproject/?_test=beta`. Last visited January 2013.

[Owen et al., 2011] Owen, S., Anil, R., Dunning, T., Friedman, E.: Mahout in action. Manning Publications Co. (2011).

[Picard, 2000] Picard, R.: Affective computing. The MIT Press (2000).

[Santos and Boticario, 2012] Santos, O., Boticario, J.: "Affective issues in semantic educational recommender systems". In Proceedings of the 2nd Workshop on Recommender Systems for Technology Enhanced Learning (RecSysTEL 2012). Manouselis, N., Draschler, H., Verber, K., and Santos, OC (Eds.). Published by CEUR Workshop Proceedings (2012), 71–82.

[Stein et al., 2012] Stein, S., Ware, J., Laboy, J., Schaffer, H. E.: "Improving K-12 pedagogy via a Cloud designed for education". International Journal of Information Management (2012).

[Vaquero et al., 2009] Vaquero, L. M., Rodero-merino, L., Caceres, J., Lindner, M.: "A Break in the Clouds: Towards a Cloud Definition". ACM SIGCOMM Computer Communication Review, 39, 1 (2009):50–55.

[Verbert et al., 2012] Verbert, K., Manouselis, N., Ochoa, X., Wolpers, M., Drachsler, H., Bosnic, I., Duval, E.: "Context-aware Recommender Systems for Learning: a Survey and Future Challenges". To appear in IEE Transactions on Learning Technologies (2012).