CLAT: Collaborative Learning Adaptive Tutor *

Alaeddin M.H Alawawdeh, César Andrés and Luis Llana
Departamento Sistemas Informáticos y Computación
Universidad Complutense de Madrid
E-28040 Madrid. Spain.
e-mail: alaeddin@fdi.ucm.es, c.andres@fdi.ucm.es and llana@sip.ucm.es

Abstract

In this paper we introduce CLAT, a multi user online e-learning tutor. This system is used to improve the knowledge of the students in different environments. A novel algorithm to share the knowledge is presented. It is based on letting the students to share their experience with the rest of their partners. In order to do this task, CLAT provides a dynamical structure that is updated under some constrains. In particular, CLAT adapts its behavior not only individually for each student but also by considering the performance of similar students. The core of its adaptive part is based on the classification of students into classes (groups of students sharing some attributes). By doing that, the past behavior of students of the same class determines how CLAT interacts, in the future, with students of that class. That is, CLAT learns how to deal with each type of student.

Keywords: Intelligent Tutors, Education

1 Introduction

Online courses [4, 6, 1] and collective knowledge [7] approaches uncover several contradictory findings about the best way to conduct teaching and learning online. The importance of recognizing the feelings, reactions, and responses of the students in an online environment is really important, but it is a hard task. Most instructors or designers simply move text-based courses to the Internet following some form of pedagogy and there are very few studies that examine this phenomenon. But, there are several promising techniques focusing on this goal based on extracting structured data from unstructured user contributions [2, 3].

In this paper, a Collaborative Learning Adaptive Tutor is presented CLAT. This system let users to interact with a set of tests, in order to increase their degree of knowledge. CLAT has been developed using software engineering principles, and make use of formal methods in the testing module. On the one hand, generally speaking, software engineering can be considered as a systematic and disciplined approach to develop software. It concerns all the aspects of the production cycle of software systems and requires expertise, in particular, in data management, design and algorithm paradigms, programming languages, and human-computer interfaces. It also demands an understanding of and appreciation for systematic design processes, non-functional properties, and large integrated systems. On the other hand formal methods refer to techniques based on mathematics for the specification, development, and verification of software and hardware systems. The use of formal methods is especially important in reliable systems where, due to safety and security reasons, it is important to ensure that errors are not included during the development process. Formal methods are particularly effective when used early in the development process, at the requirements and specification levels, but can be used for a completely formal development of a system. It has been argued, usually with very good arguments, both that formal methods are very appropriate to guide the development of systems and that, in practice, they are useless since software development teams are usually not very knowledgeable of formal methods in general, and have no knowledge at all of what academia is currently developing. It is interesting to remark that some of the strongest advocates of formal methods have almost “no experience on the development of real systems”. Thus, one of the main goal of this paper is to present CLAT as a formal real system development for e-learning.

Next we briefly sketch the main characteristics of CLAT. First, CLAT combines individual profiles for each student with general profiles for each class of students. By doing so, CLAT is able to adapt its behavior with respect to a student not only individually, on the basis of her previous fails and successes, but also by taking into account the performance of the rest of students (current students as well as former students). That is, the system learns how to interact with a student by adapting the general profile of her class(es) to
her necessities. However, the progress of the users while experimenting with the system is individually controlled. CLAT keeps profiles for each student where all the relevant information (from her previous sessions) is recorded. In particular, the system points to the next topic that the user should explore once she has reached a certain command in the current topic. Another important feature of CLAT is that it can automatically generate based on the student’s skill. We consider that CLAT may increase the success rate of students in two ways. On the one hand, students can regularly check their progress by self-evaluation. On the other hand, teachers can find out which parts of the course are more difficult for the students (CLAT provides the teacher with a private interface that allows her to access the information about the performance of students). An additional contribution of CLAT is a module implemented to verify the identity of the students. This module makes use of Bayesian text filtering to determine the similarity between different executions of the same student.

The rest of the paper is structured as follows. In Section 2 we present the initial structure of CLAT. In Section 3 we present the idea of how to identify the users that are filling the tests. Following, in Section 4 we discuss different features and measures of CLAT. In Section 5 it is presented our system in a collective user scenario solving some features previously discussed. Finally, in Section 6 we give our conclusions and some lines for future work.

2 CLAT structure

In this section the initial structure of CLAT is presented. Let us note that the main advantage of using an intelligent tutor with adaptive capabilities is that it can be automatically adapted to the students. CLAT can adapt its behavior not only on the basis of the fails and successes of the current student, but also on the experiences of the rest of students.

In CLAT, at the same time, $n$ different users can be connected. Previous interactions of the users is recorded on the server. This information is used for obtaining statistical data and for classifying the users in different statements. When a user asks a new test to the server, by sending Do test request, the server sends to this user Send test $(x)$, where $x$ is the test to be solved. After receiving the test, the user sends the answer to the server, and according with the stored results, it will consider it as a good answer or not.

CLAT implements a user classification. This distribution not only depends on the answers provided from the users on different topics tests, but also in the effort provide from each user to generate new tests.

Each group is a set of sorted users (according to the amount of gained points). So, the set of guru users is made of the $u_1$ best users of the ranking, the set of expert users is made of the next $u_2$ users, and so on. As it usually happens in knowledge communities, the amount of users in each class should follow a pyramidal structure. Thus, the condition $|u_1| < |u_2| < \ldots < |u_m|$ will be considered. In Figure 1 a basic users structure is presented in CLAT. There are three different levels, which are Experts (E), Medium (M), and Basic (B). Also, in this Figure the values 0, ..., 4 represent tests suites.

Any new user starts solving tests at point 0, which is represented in the base of the system. If the user passes this initial tests suite, then she will be included into the (B) level.

If users want to continue being in their current level, then they have to perform some tests of this level, that means they are being “updated with the current knowledge” of this level. These tests suites are represented in Figure 1 by number 1, and usually they are proportionated by the other users belonging to this level, or the users of a higher level (denoted by 2). An user can upgrade to a higher level if she performs correctly the tests suite 3, or downgrade if she is unable to answer the tests suite 4 correctly.

Taking into account the user structure presented in Figure 1, then in Figure 2 we present our ideal structure evolution. In this structure is presented how all the users of the system are becoming into experts. At the beginning the most users of the system are in the (B) level, and at the end the most users are in the (E) level. It means, at the end, all new knowledge is spread out into the users of the system.

3 The Validation of a Student ID

In order to ensure a personalize treatment, students access the system through a login page. This allows the system to recover the data from previous sessions. At the beginning of the course, students are provided with a pass-

![Figure 1. Users structure.](image-url)
word. They log in by giving their ID-numbers and the password. This mechanism tries to avoid attacks to previous sessions of students. For example, an attacker could ask the system for previous exercises and provide wrong answers. Then, when the real student logs in, she will find out that the system thinks that she did not understand the concepts covered in previous sessions.

So, to solve this problem, in CLAT we have implemented a novel approach to detect unexpected behaviours of the users. Next we present the main steps of this module and next the implementation details. First of all, we have to define what is an unexpected behaviour. Basically, at the beginning of each session the first set of tests that the user answers always contains some tests that she answered in the past. Taking into account that all the interactions of the users and the system are collected in our database, then we can compare what the student answered and what she is answering right now. CLAT assumes that student learns the subject, so it expect that some questions that had a wrong answer in a first iteration with the student, they are answered correctly answered in a second iteration. However, during the implantation phase of CLAT, we detected that there were some situations that happened in CLAT that forced us to adapt this validator in order to avoid them. Following we report a situation that show unexpected behaviours. “Some students always use some common expressions in their answers. We detected that some of them always wrongly wrote these expressions (the verb, the subject...). CLAT detected a wrong behaviour because of during a session one of this student always wrote correctly the expressions and the session recorded in the following day she continued writing wrongly the expressions. Thus, there was another person answering her tests.”

A typical execution of our this module is following presented. During the rest of the section we will denote by $\rho$ the test that is being compared. The first thing that is extracted from a test is the language, so that we can differentiate, in our case, between English, German, French and Spanish. Next the representation of $\rho$ is analyzed. All the information is extracted from the $\rho$ while empty words, that is, words that are not representative of $\rho$, are removed by using a stop list (typically, this list includes words such as “a”, “the”, “is”, “etc”, etc). As expected, these lists vary for different languages. Next, $\rho$ is tokenized so that the frequency on the test of each relevant term is computed. These terms are calculated by using a list of inverted terms (we will deal later with this concept). Given a word $i$ and a document $j$, the weight of the word is computed by using the following expression:

$$wd_{ji} = \text{Freq}_{ji} \times \log_2 \left( \frac{\text{NTD}}{\text{NDA}_i} \right)$$

where $\text{Freq}_{ij}$ is the frequency of $i$ in document $j$, $\text{NTD}$ is the total number of documents on the case database for this user and $\text{NDA}_i$ is the number of documents that contain the word $w$. In Table 1 we show how the relevant information contained in each test is recorded by using events (contained all the information) and cases (containing the information relevant for further processing).

Once all the weights for $\rho$ have been computed, the similarity between this and other documents classified in the case database is studied. For the classification of cases, CLAT is based on a list of inverted terms to store all the terms that appear in previous cases as well as a list containing the document numbers that have that word among its list

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subject</td>
<td>The name of the subject</td>
</tr>
<tr>
<td>Date</td>
<td>Date of dispatch</td>
</tr>
<tr>
<td>Origin</td>
<td>The IP number and the operative system</td>
</tr>
<tr>
<td>Message</td>
<td>The answers for the test</td>
</tr>
<tr>
<td>Error</td>
<td>Indicates whether it is an unexpected behaviour or not</td>
</tr>
<tr>
<td>Terms</td>
<td>Terms extracted from the Message and the Subject</td>
</tr>
<tr>
<td>Weights</td>
<td>Weights associated with each term of the test</td>
</tr>
<tr>
<td>Document number</td>
<td>Document number in the case database</td>
</tr>
<tr>
<td>Language</td>
<td>Language used in this test</td>
</tr>
</tbody>
</table>

Table 1. Structure of an event and of a test.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total number of occurrences</td>
<td>Total number of occurrences of the word among all documents</td>
</tr>
<tr>
<td>Vector of occurrences</td>
<td>Vector with the number of occurrences of the word in each document</td>
</tr>
</tbody>
</table>

Table 2. Structure of the list of inverted terms and of its elements.
of terms. The list of inverted terms has been implemented with a hash table, having as keys in this table are the new terms. The values of the table are elements with the structure defined in the bottom of Table 2. Finally, the similarity between a document of the database $j$ and $\rho$ is computed as

$$\text{sim}_{j\rho} = \frac{\sum_{i=1}^{m} wd_{ji} \ast wd_{pi}}{\sqrt{\sum_{i=1}^{m} wd_{ji}^2 \ast \sum_{i=1}^{m} wd_{pi}^2}}$$

where $wd_{xy}$ is computed as stated before, denoting the weight of word $y$ in document $x$, and $m$ is the number of weight vectors.

Once we have all the similarities, we choose the three documents providing the highest similarity with $\rho$. If the highest one is greater than 65% of the addition of the other two, then we take the result corresponding to the highest. Otherwise, we make a majority voting among these three cases. Once $\rho$ is processed, and a verdict regarding whether it contains an unexpected behaviour has been reached, then $\rho$ is added to the case database so that it can be used for analyzing forthcoming tests. Let us remark that with this module, CLAT allows us to detect the unexpected behaviours of the students.

### 4 Security and statistical measures on CLAT

In order to guarantee the fault absence, a methodology of performing testing security task is provided in CLAT. There are several ways to classify testing techniques. In this work, there are used passive testing techniques, which are non-intrusive with respect to the system under test, and the security properties can be represented by using formal syntax. Passive testing do not use any set of test in order to provide a verdict of how good a system is. This technique uses the log of the system in order to recognize erroneous behaviors. We have detected within our architecture several problems. When users are interacting with the system, all actions performed from them are stored in a global log, and the measure of this log is huge.

The global log structure presented in CLAT is presented in Figure 3. There, all messages sent from users are recorded into a file. We can express security properties in order to check the correctness of the log. For example, let us consider the following property: “CLAT implements a FIFO user policy”. That is, if the user $C_1$ asks for a test before $C_2, C_3, \ldots$ then, the server will provide firstly to $C_1$ the test to be solved. According to the global log presented in Figure 3, $C_1$ asks for Do Test before $C_2$, so the server will answer with Send Test to $C_1$ before to $C_2$. Let us note, than before receiving $C_2$ its notification, a new user, by means $C_n$, asks for a new test. Then, according to the previous rule, the answer to $C_2$ should obtain the test before than $C_n$.

Let us consider another property presented in CLAT. As it was mention, CLAT provides a methodology to perform different tests in different topics. Sometimes it is necessary that previous knowledge should be learned before advancing in tests. Let us consider the test structure presented in Figure 4. This structure represents that the initial test to be performed by an user is Test 0, and performing this she will be able to answer or Test 1, or Test 2, or Test 3. The arrows from Test $i$ imply which tests are opened after correctly performs the test $i$. By using passive testing techniques, we are allowed to express properties such as: “At the beginning only Test 0 can be performed”, or “If the Test 6 is performed then previously Test 1 and Test 2 were correctly performed”.

The other main features on CLAT is possibility to to get, in an easy way, statistical information of the users. This information is used to provide a good classification of students. This information should be able at the end of each day, and it is extracted from the interactions log of the users recorded in one day. Each day the information has to be mixed with previous information, and a new user hierarchy is provided.

There are three problems when we perform these tasks in the structure presented in Section 2. The first problem
is the huge size of the log presented in the server. Let us consider that the amount of users, with respect to time that are connected is denoted by \( N(t) \), and the average of the amount of actions of any user with the system is \( K \). Then we have that an usual size of log between two timed values \( t_1 \) and \( t_2 \) is represented as:

\[
\int_{t_1}^{t_2} N(t) \, dt \cdot K
\]  

(1)

The second problem is relative to performed measures. When we perform \( p \) different properties by using passive testing then we have that the complexity of performing this task is presented as:

\[
p \cdot \int_{t_1}^{t_2} N(t) \, dt \cdot K
\]  

(2)

The third problem is relative to perform the statistical information. In order to provide \( m \) statistical information, we have:

\[
m \cdot \int_{t_1}^{t_2} N(t) \, dt \cdot K
\]  

(3)

Let us note that in order to increase the performance of CLAT, we should reduce the values of equations 1, 2 and 3.

5 CLAT in a collective user environment

In this section we present how to reduce the previous problems in CLAT to work in a collective user environment. First, we focus on the size of the log. The previous approach assumes to have a big log, where all user interactions with the server are recorded. In this update, the software performed by users increase the functionality, in order to reduce the memory used in the server. All client software, will be able to store all interactions of the incoming user, and after will process this information, and will send a resume to the server. This idea is presented in Figure 5. When an user finishes interacting with the system, then the passive testing techniques will be used with the local stored data, and will send this information to the server. Let us note, that if we only check security properties in alone nodes then the size of the server for storing the log is null, and the size of the users is \( K \) (Similar value than Equation 1).

But, not all properties can be checked in users software. For example, the FIFO property must be checked, the information recorded in users is not enough. The information regarding when the user \( C_i \) and the user \( C_j \) connect to the system is only provided in the server. Let us assume that we have \( p \) properties, where \( q \), with \( q \leq p \) are properties that only can be checked in the server. Let \( K \) be the average of actions performed by users (see Equation 1), and let \( K' \), with \( K' \leq K \) be the actions that have to be recorded in order to check the \( q \) properties. Then, the amount of data recorded in server are:

\[
\int_{t_1}^{t_2} N(t) \, dt \cdot K'
\]  

(4)

Let us note, that the log size in users have not be reduced, because very often the properties \( p-q \) that matches the local log need to check the complete interaction log.

The second problem that we focus in this update is the time to check all properties. Let us note that in previous approach, we have a big log, recorded in the server, and all interaction were recorded on it. The time associated to check the properties were presented in Equation 2. By using this new architecture paradigm, we reduce the computation. The passive testing tasks are performed now not only in the server, but also in the \( n \) users. The task of performing in users can be done at the same time, because they are in different machines. So that, the time associated with this task is:

\[
q \cdot \int_{t_1}^{t_2} (N(t) \, dt \cdot K') + (p-q) \cdot K
\]  

(5)

The last problem is the time to check the statistical properties. As we have computed previously in Equation 3 this task continues being very cost. However, this property only uses isolate user information. That is, this task can be performed with all guarantee in the user clients. So, the server only has to be able to record the results generated by each user clients, at the end of all interactions. The new equation to compute the \( m \) statistical information is:

\[
m \cdot K
\]  

(6)

This cost is computed in each client, and it does not perform any disadvantage in the server. Next, all information about the statistics of the user are sent to the server. Let us remark another feature presented in this update. In Figure 5 is presented two calls to the server: Statistical request and Statistical response. This input/output action respectively are used in order to compute.
two stages. The first stage is focused on classifying users into system; and the second one is used to create/modify/update the tree structure of the topics.

6 Conclusions and Future work

In this work we have presented our tool CLAT. This tool has been developed focusing on solving the problems of testing task and synthesizing statistic information. Both tasks are traditionally performed in the server. We suggest to increase the power of client software, in order to reduce the computational time presented in the server. Furthermore, we have proved that by using the CLAT architecture, the amount of interactions of the users that has to be mixed in logs, in order to perform the testing task, is reduced. In addition we have presented a novel module, based on Bayesian text filter to identify the ID of the users.

It is worth to point out that a very important part of any tutoring system is the feedback from the users. While designing our system, we have been specially careful at this point. For instance, let us consider the answer of our system after a test is made. If the student provides the right answer then the system returns a congratulations message. The difficulties start when managing wrong answers. The easy solution consists in notifying that the answer was wrong and provide the right answer. In this scenario, the student will try to understand what she was doing wrong by pattern matching. We consider that this is not the best practice. We have preferred to return a suggestion about what the student should do (indicating what the error was) instead of giving the right answer.

We have also paid special attention to avoid cheating. As it is pointed out for example in [5], some students tend to learn how to cheat the system instead of learning the current contents. We do not claim that our system is totally fool-proof (actually, we do not think so!) but we have tried to detect some funny answers. For instance, if we ask for the value of 3+4 a student may answer 5+2 (non so trivial examples include the application of higher order functions in an unexpected way). Actually, this is a right answer, but it is not what it is expected. If CLAT detects such a right answer, it will indicate that it is correct but it will ask for the most correct answer. Finally, even though the management of answers has been a specially hard part to develop, we think that the effort has been worth. Firstly, students will see their mistakes and try to correct them. Secondly, they will be soon convinced (we hope) that it is senseless to spend time trying to fool the system.

Students will be also allowed to ask for hints. The type of hints, that the system provides, depends on the number of hints the student has already asked for in the current exercise. For instance, if the student has provided a wrong answer, a first hint will only provide a message saying which type of error it was, that is, whether it was a syntactic error, a type error, or a semantic error. Afterwards, in case the student needs more hints, the error will be explained more precisely. Finally, if a student is not able to provide the right answer, she can press the give up button and the answer will be presented.

Thus, as future work, we would like on the one hand to include more intuitive answers in each test, and on the other hand to implement new situations to detect unexpected behaviours, such as using probabilities and stochastic information.

References


