An Architecture for E-Learning System with Computational Intelligence

Marc El Alami, Nicolas Casel, and Denis Zampunieris

CICeL, Faculty of Sciences, Technology & Communication, University of Luxembourg, 6 rue Richard Coudenhove-Kalergi, L-1359 Luxembourg-Kirchberg, Grand-Duchy of Luxembourg
{marc.elalami, nicolas.casel, denis.zampunieris}@uni.lu
˜http://cicel.uni.lu

Abstract. In this paper, we introduce a new kind of software tools intended to offer a virtual educational and/or training environment online: proactive e-Learning Management Systems.
These computational intelligence based systems are designed to improve the users' online (inter)actions by providing programmable, automatic and continuous intelligent analyses of the users' behaviors, augmented with appropriate actions initiated by the LMS itself.
We show how we implemented the proactive part of our LMS on the basis of a dynamic rules-based expert system. We also sketch how it looks like from a user's point of view. Finally, We give some examples of intelligent analysis of users' behaviors coded into proactive rules.

Key words: Learning Management System, Proactive Computing, Rule-Based Expert System

1 Introduction

Learning Management Systems (LMS) or e-learning platforms are dedicated software tools intended to offer a virtual educational and/or online training environment. Unfortunately, current LMS are fundamentally limited tools. Indeed, they are only reactive software developed like classical, user-action oriented software. These tools wait for an instruction and then react to the user request.

Students using these online systems could imagine and hope for more help and assistance tools, based on an intelligent analysis of their (lack of) actions. LMS should tend to offer some personal, immediate and appropriate support like teachers do in classrooms.

In this paper, we introduce a new kind of LMS: a proactive e-Learning Management System, designed to improve the users' online (inter)actions by providing programmable, automatic and continuous intelligent analyses of the users' behaviors, augmented with appropriate actions initiated by the LMS itself.

Our proactive LMS can, for example, automatically and continuously help and take care of e-learners with respect to previously defined procedures rules, and even flag other users, like e-tutors, if something wrong is detected in their

behaviors; it can also automatically verify that awaited behaviors of e-users have been carried out, and it can react if these actions did not happen.

In the following text, we show how we implemented the proactive part of our LMS on the basis of a dynamic rule-based expert system, and we sketch how it looks like from a user's point of view. Finally, we give some examples of computational intelligence coded into proactive rules.

2 User Interface

Recent works (see [1] for example) also propose how to improve current web-based educational systems by adding intelligence in these systems, but these intelligent add-ons modules are as static as the initial LMS was. Indeed, they still need a click or an action from the user to activate it. Our goal was to design and develop a LMS, that is able to analyze a situation and to act spontaneously wrt. this situation without queries from its environment.
In our system, the user can receive information, help or hints sent by the proactive system at any time and with no actions needed from him. These messages should not disturb him in his work, that's why the interface has been thought in such a way that the information will be viewable at any time and in any context in the LMS.

Fig. 1. The Message Zone

A messages zone has been dedicated in the header. This alert zone is a Flash application which is able to display server messages in real-time. Messages are following each other vertically and are colored differently according to their importance.

By clicking on a message, the user opens the Messages Manager and s/he can read more details on her/his alerts and save them.

Fig. 2. The Message Manager

3 Proactive Computing

Proactive systems (see [2], [3]) act on their own initiative on behalf of the user and without waiting for users explicit command. In other words, the user is "above the loop", that means that the system can manage himself with no human interactions. The user is here to supervise the system and take last decisions. These systems are more reactive to extern events and can communicate automatically with users using the usual ways of messaging like popup, emails, alerts zone or logs for administrators. The continuous computing of users (inter)actions is possible thanks to a set of programmed behaviours augmented with appropriated actions.

The aim of proactive LMS is to help users to better interact online in a learning environment. Guiding new user through the LMS is a common problem: dynamic help will be more efficient than an elementary static interface. Moreover, proactivity is useful in case of urgent situation: it enables focusing and dealing immediately with relevant information. For instance, a learner may have difficulties with an online exercise. Our system catches, analyses and transmits pertinent data to the tutor who can quickly respond. With standard LMS, the same piece of information is merely stored in a database.

How does it work? The proactive part of the LMS is based on few principles:

- Every proactive behaviours is coded in the Rules Running System (RRS)
- Rules are pushed in a FIFO queue
- The RRS is activated for a time frequency F and performs the N first rules of the FIFO queue
- Once executed, a rule is deleted from the system
- To regenerate, a rule should clone itself to be pushed again in the FIFO queue and be reactivated later

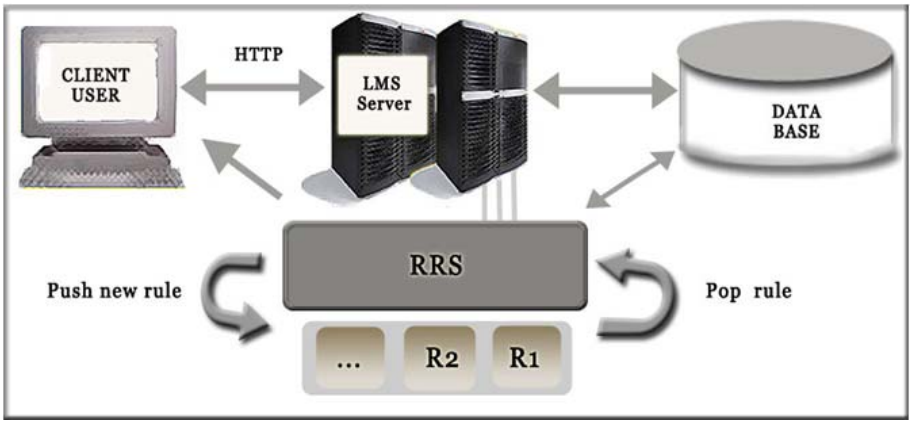

Fig. 3. The Rules Running System (RRS)

Figure 3 describes the different actors interfering in the system. The RRS is the extra element that wouldnt appear in a traditional LMS architecture. It is independent from the initial LMS server but works in complementarity with it by sharing the same database. The RRS also directly communicates with the Client and the LMS server. For more details, please see [4].

A rule is made of five parts: data acquisition, activation guards, conditions, actions and rules generation.

```
data acquisition :
        es = get_user(S)
        ec = get_course(C)
activation guards :
        es.isConnectedToCourse(C) == true
conditions :
actions :
        showMessageBox(es.session, "Welcome to the course " + ec.name)
        showMessageBox(es.session, "Do not forget to take a look at the forum "
                + "dedicated to the course " + ec.name + " called " + ec.forum)
rules generation :
        if (activated == false)
        then cloneRule(self)
```

Fig. 4. The Algorithm to Run the Welcome Rule

The **data acquisition** allows a rule to get information from the LMS in order to use these data in its others parts. The second part (**activation guards**) is made of a set of tests that determine wether or not the rest of the rule will be processed. If the activation guards are evaluated positively, the conditions and actions parts are performed. Otherwise, these two parts are ignored and

the rules generation is executed. The third part (**conditions**) is a set of AND-connected tests on local variables that, once evaluated, determine if the action part will be executed afterwards. Syntax and semantics of conditions tests are equivalent to activation guards tests. The fourth part (**actions**) is made of a list of instructions that will be performed in sequence. The fifth and last part (**rules generation**) is realised at the end. It allows the rule to regenerate itself or others rules that will be pushed in the FIFO queue and executed later.

4 RRS Architecture

Before starting to develop the system, some technological choices were made:

- The system should be simple, effective and fast: therefore we choose the C language that has these advantages.
- The LMS is able to send information to the client at anytime without waiting for a client query. Indeed, using sockets was clearly necessary in order to keep a permanent connexion where both parts can be servers.
- The client is able to treat and post text in a browser automatically without refreshing: Flash is the right client application to meet these requirements. Widespread (97% of the browsers) and crossplatform, it can be coupled with several servers technologies (Java, JSP, PHP, ASP, etc) and the most important property is that it can handle a socket connexion.

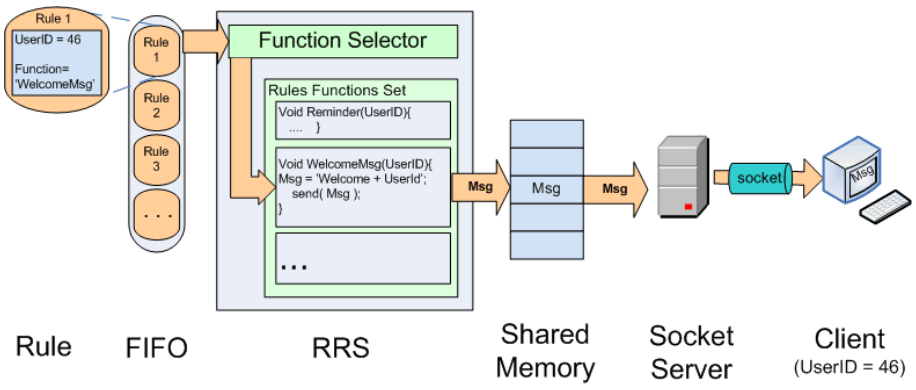

Fig. 5. The RRS Architecture

Figure 5 shows an overview of the Rules Running System. The RRS is able to pop a rule from the FIFO queue, check the type of this rule and parameters included. The reference to the function (or proactive behaviour) which performs the rule is given to the RRS. Actually, the RRS accesses a library that defines

every rule and instructions to execute it. For instance, these actions can be sending a message to the client through the socket connexion or by email. In this example, the rule is to send a welcome message to the user n°46. The message is build in the RRS and put in the shared memory. The server checks the shared memory and post the welcome message to the right online user. The use of the shared memory allows the RRS and the server to work asynchronously. The RRS can continue to handle new rules while the server is in charge of sending the message to the client.

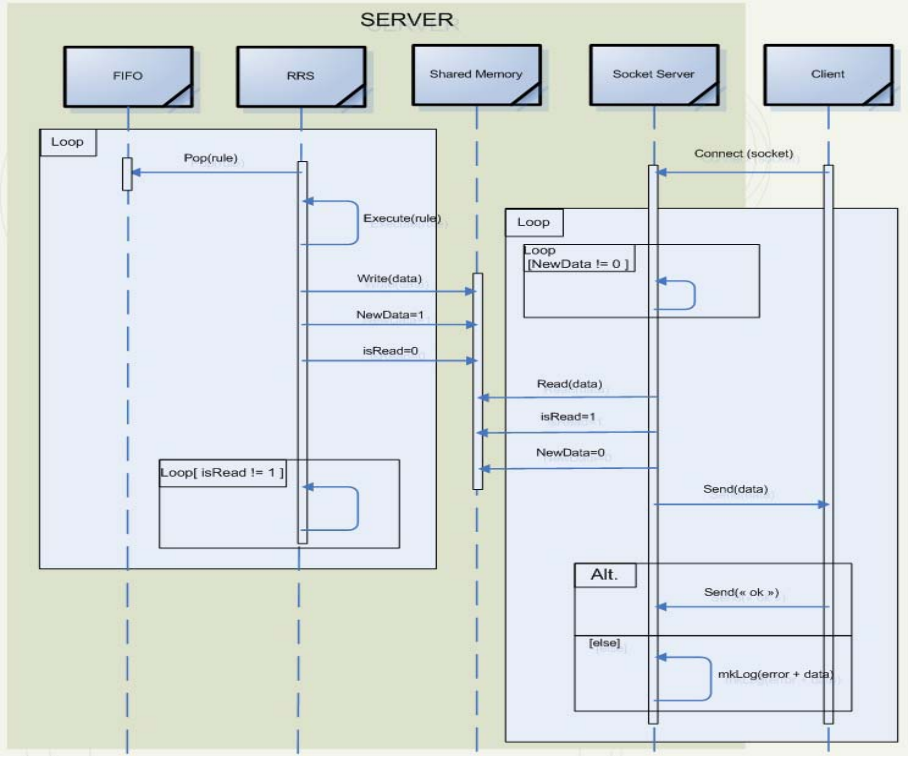

Fig. 6. The Message Sending Sequence Diagram

Figure 6 shows the path followed by a message from the state of a rule on server-side to the print on the client screen. First, the RRS pops the rule from the queue, tests the activation guards (ex: test if the user is connected) and executes the rule. The message is build and put in a shared memory where the socket server can easily access and read it. Before reading the shared memory , the server checks if there is a new message to send. Meanwhile, a client is connected and if the message concerns him, the server will use the open socket to

address him the data read from the shared memory. Finally the client will send back a positive acknowledgment. If there is no response from the client after a given time, the server creates a log to inform the administrator. The RRS waits for the socket server to read the message in the shared memory before continuing.

With the continuous evaluation of large sets of rules, an efficiency problem might appear with the system saturation. Indeed, as every rule makes an average of two to three requests to the LMS database, the mean reaction time of the LMS to a user request is increased, sometimes in a severe way. This problem can be solved with the introduction of "lazy evaluation" in the main algorithm. It is an optimization technique that attempts to delay computation of expressions until the results of the computation are known to be needed. For the interested reader, more information about this solution can be found in [5].

5 Rules for Intelligent Analysis

These proactive behaviours coded into rules are the bases for an automatic and intelligent analysis for the current state of the whole system and of its changes implied by user actions and interactions.

```
data acquisition :
        es = get_user(S)
        et = get_user(T)
        ec = get_course(C)
        date = get_date()
activation guards :
        date > ( ec.startDate + 7 days)
conditions :
        es.numberOfConnections(ec.forum) == 0
actions :
        sendLMSeMail(to = et.name, subject = "Warning", data = "e-student "
                + es.name + " did not use the forum " + ec.forum.name
                + " after one week... please check with her/him")
rules generation :
        if (activated == false)
        then cloneRule(self)
```

Fig. 7. Algorithm to Run the Forum-Warning Rule

Figure 7 represents the structure of a specific rule that is intended to check that the e-student S used at least once the LMS forum communication tool dedicated to the e-course C one week after his registration to that e-course and if not, to notify it by an LMS email to the e-tutor T so that he can check with the e-student what is the problem. In the data acquisition, the system gets all the information needed (information about the student, the tutor, the course and the current date), then the activation guards are performed to determine if the rule should be activated or not; if yes, the rule enters the conditions part

that determine if the rule should be processed. In that case the rule enter the actions part and accomplishes a list of instructions (in our case, the instruction is to send an email to the tutor with all the information). The fifth and last part (rules generation) allows to regenerate rules that have not been activated.

6 Conclusion

Current Learning Management Systems are fundamentally limited software tools: they are only reactive, user-action oriented software. These tools wait for an instruction and then react to the user request. They do not offer some personal, immediate and appropriate support like teachers do in classrooms.

We introduced a new kind of Learning Management System: proactive LMS. These e-learning platforms with computational intelligence, are designed to improve their users' online interactions by providing programmable, automatic and continuous analyses of users actions augmented with appropriate actions initiated by the LMS itself.

We showed how to implement such a proactive LMS on the basis of a dynamic rules-based expert system. We also gave some examples of proactive rules declarations. Finally, we sketched how it looks like from the users' point of view.

Future work includes the design and the implementation of sets of rules (packages) dedicated to common users needs, as well as to elaborated intelligent analysis coded into proactive rules.

These sets of rules will be of two kinds: standard packages that one will be able to use as is, as well as abstract packages (templates) that one will have to tailor to its specific needs by using appropriate tools.

References

1. Brusilovsky, P. (2003) "A component-based distributed architecture for adaptive Web-based education". In: U. Hoppe, F. Vardejo and J. Kay (eds.) Artificial Intelligence in Education: Shaping the Future of Learning through Intelligent Technologies. (Proceedings of AI-ED'2004, Sydney, Australia, July 20-24, 2004) Amsterdam: OIS Press, pp. 386-388.
2. D. Tennenhouse, "Proactive computing", Communications of the ACM, 43, 5 (2000), pp. 43-50.
3. A. Salovaara and A. Oulasvirta, "Six modes of proactive resource management: A user-centric typology for proactive behaviors", Proceedings of the Nordic conference on human-computer interaction, ACM international conference proceedings series, 82 (2004), pp. 57-60.
4. D. Zampunieris: "Implementation of a proactive learning management system", Proceedings of E-Learn World Conference on E-Learning in Corporate, Government, Healthcare, & Higher Education, Hawaii, USA, October 2006, Eds. Th. C. Reeves and Sh. F. Yamashita, ISBN 1-880094-60-6.
5. D. Zampunieris: "Implementation of efficient proactive computing using lazy evaluation in a learning management system", m-ICTE - International Conference on Multimedia and Information and Communication Technologies in Education, Seville;Spain, 2006.