STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LVII**, Number 4, 2012

BUILDING INTELLIGENT KNOWLEDGE MANAGEMENT SYSTEMS

OVIDIU PÂRVU AND SANDA M. DRAGOŞ

Abstract. Knowledge management systems are used in most of the large companies and institutions throughout the world. Currently, most of them are not designed to adapt to the dynamic changes of the surrounding environment. Intelligent plugins can be integrated in these systems in order to increase their reactivity and adaptability. In the present paper, we describe an intelligent plugin for a knowledge management system (KMS) used for managing the PhD theses in the Babes-Bolyai University. The KMS which uses our plugin is called Doctorate Theses Administration Platform (DTAP) [9, 10]. Our plugin uses an intelligent agent in order to help users go through all the required steps of handling a thesis. The behaviour of the application changes dynamically based on the interaction with the users. Results show that both the time needed to manage the theses and number of encountered errors were reduced considerably after the integration of the above mentioned components.

1. Introduction

The term *Knowledge Management System* has many definitions in the literature, but in the present paper we will use only the meaning indicated in [2, 11]: "The KMS represents a class of information systems (ISs) applied to managing organizational knowledge". Simply stated, these systems are used for managing the knowledge of an organization. The collective knowledge of the organization is stored either in a repository or in an archive of documents. At micro level everyone in the organization is contributing to this database, so it can be seen as a dynamic entity which is continuously evolving over

Received by the editors: November 12, 2012.

2010 *Mathematics Subject Classification.* 68T05.

1998 *CR Categories and Descriptors.* J.1 [**Computer Applications**]: Administrative Data Processing – *Education*; I.2.1 [**Computing Methodologies**]: Artificial Intelligence – *Applications and Expert Systems*; H.3.4 [**Information Systems**]: Information Storage and Retrieval – *Knowledge Management Systems and Expert Systems*.

Key words and phrases. knowledge management system, education, reinforcement learning, templating system.

time. A KMS eases the management of the information by using its built in mechanisms.

According to [1, 2] the most important functions of such a system are: *creation*, *storage* and *retrieval*, *transfer* and *application of knowledge*.

The *creation* of knowledge is accomplished by developing new knowledge or replacing existing one from the repository. The KMS includes a repository which allows sharing, creating, amplifying and enlarging the organizational knowledge [2]. Everyone has to collaborate and participate to the contents of the database in order for it to expand. Whether it is a document, a procedure or a new standard, it is important for it to be integrated in the system which every colleague can access.

This brings us to the second function of *storing* (and *retrieving*) data from the repository of the KMS. These are the basic operations which must be carried out in order for the information exchange to take place. Simple file sharing is not enough and specific tools which aid collaboration should be provided. These tools can include database interrogation forms, a logging system which records everything that has been done and maybe also small software modules which have an intelligent tutor integrated. Using these tools, even inexperienced employees can retrieve information from the database in a format which is easy to understand. The ones who share or store information must find the experience both professionally and personally rewarding, such that they remain motivated to use the system [12]. Some companies use rewarding systems to motivate experienced employees to take the time to document their work even if they may not ever need to make use of such knowledge. In other companies, it is part of the job description to write small tutorials and guides, such that everyone has to do it. There are also companies in which every modification is logged, such that history records are generated automatically and can be used as samples for good/bad practices.

A KMS, as its name suggests, is not about exchanging information, but knowledge. It is clear why there is a need for the third function which is the *transfer* of knowledge between employees. In order for the exchange to be done in an optimal way, the system must provide an efficient communication support not only between individuals, but also between different departments within the same company. For instance, employees from one department may post a question on a forum and colleagues from all over the company may view and answer this question. Another useful utility is one that allows employees to check the changes done by other colleagues.

The aim of a KMS is that employees collaborate and *apply* in practice the gained knowledge in order to obtain better results than before.

2. Doctorate Theses Administration Platform

These mechanisms of a KMS can be applied when designing an application for managing the PhD theses in a university. We implemented such an application for the Babeş-Bolyai. It is called the Doctorate Theses Administration Platform and we will refer to it as DTAP in the rest of the paper. Previous to DTAP, the employees of the Institute of Doctoral Studies (IDS) were using obsolete and time consuming methods for managing PhD theses. This led to the implementation of this system. First of all, the information regarding the PhD students was archived on paper. Therefore, searching for a specific thesis or backing up the entire archive implied a lot of work. Secondly, all the documents required by the Romanian Educational system were filled in manually by the employees of the IDS using Microsoft Word templates. This was both time consuming and error prone. Finally, the information which had to be made public was written by hand for every thesis. Considering that the information was displayed taking into account different sorting criteria (e.g. title, author's name etc.), the information had to be duplicated for each one of these sorting criteria. DTAP addresses these issues by using the mechanisms described next.

First of all, it stores all the information regarding the PhD candidates into a database. Therefore, backing up the entire information implies backing up the database, which is not a difficult operation when using a database management system. Moreover, all the data which needs to be made public does not have to be written by hand anymore. Queries could retrieve the needed information from the database considering the sorting criteria chosen by the user. DTAP also includes a tool for generating reports as ".xls" (Microsoft Excel) files. These are very useful to the employees of IDS, because they must create annual reports regarding the performance of the University and its staff (e.g. Supervisor which coordinated most PhD theses in 2012).

Using the data regarding the PhD candidates and a set of templates stored in the database, the employees of the IDS can generate automatically all the documents required by the Romanian Ministry of Education (i.e. Adresă minister, Anunţ susţinere, Decizia rectorului, Listă documente, Proces verbal, Referat preliminar, Referat final). The procedure of generating the documents can be split in two steps. The first step is retrieving the required templates from the database. The second step is to parse the templates, replace variables from the template with their actual values and format certain parts of the template accordingly (e.g. bold, italic). The generated document is stored in a temporary folder until it is downloaded. Immediately after the user downloads it, the document is removed in order to reduce the memory usage. Due to lack of space the algorithm and architecture of the system are not presented

here, but more details can be found in [9, 10]. We would like to emphasize the improvement obtained from using this system. The total time needed to generate the documents for a thesis prior to DTAP was 7 * 180 seconds = 1260 seconds, whereas the total time needed to generate the documents using DTAP is 7 * 3 seconds = 21 seconds. Thus, using DTAP the employees of the IDS will need only 1.66% of the time they required before to generate all the documents.

This paper, however, presents the integration of an intelligent plugin in the KMS. The intelligent agent helps the users of DTAP by offering suggestions regarding the steps that need to be followed when managing a thesis. Suggestions aim is to prevent human errors which could delay the submission of the thesis and the PhD students viva respectively.

In section 3 and 4 we will describe other approaches of implementing intelligent systems and how our own approach is different from the existing ones. Moreover, we will give a short introduction for reinforcement learning, the method we used for implementing the intelligent plugin in DTAP.

3. Intelligent Knowledge Management Systems

According to John McCarthy Artificial Intelligence is "*the science and engineering of making intelligent machines*". In order for these machines to fulfill their task, they have to interact with the surrounding environment. Learning from the interaction with the environment underlies all theories of learning and intelligence [13]. Such intelligent systems are used for developing adaptive educational systems. In our case, we have used the reinforcement learning (RL) strategy for implementing an intelligent educational agent.

RL is a class of solutions in which the problem to be solved is defined in terms of rewards and punishments [6]. An autonomous agent will learn based on the received rewards and punishments which actions to choose in order to reach its goal. These rewards and punishments are given by a trainer depending on the problem to solve. For instance, if the agent wants to learn how to play a game, the trainer can punish him for losing, reward him for winning and offer him no payoff for simply playing [8]. The task of the agent is to learn the most optimal strategy for reaching its goal.

Single agents or multi agent implementations using RL have been used for implementing various educational systems [3]. The difference between these two types of systems is the fact that in a single agent system the learning process is influenced only by the interaction with the environment, while in the multi agent system the learning process is influenced by all the other agents present in the system as well. There are multiple examples of educational

systems which use RL in order to dynamically adapt their behavior to the changes of the environment.

Among these we would first like to mention the educational system RLATES [7]. In this case the pedagogical policy applied to one student is improved based on the experience of other students with similar characteristics. The experiments show that the system learns through a trial and error technique at the same time the students learn. One of the benefits of using RL is that it eliminates the problem of overtraining encountered at supervised learning.

On the other hand, Cordillera [5] implements a different learning strategy. Cordillera uses a method of RL which improves the efficiency of the system by inducing which policies should be used for the students. The system starts with a set of policies denoted as exploratory corpus. This exploratory corpus was collected while letting the system make random decisions while interacting with real students. The feedback received from the users determines which policy is the most appropriate one.

A different approach is presented in [4] where the end purpose of the system is to discover efficient/inefficient teaching strategies. The efficient teaching tactics can be applied by teachers in real classrooms while the inefficient ones should be avoided. This approach was used in the implementation of NormGain and InvNormGain [4] which learn what are the best and worst teaching strategies respectively through their interaction with the students.

The characteristic which all the above described software products share is the fact that they are directly addressing the needs of teachers and/or students.

4. Navibot

Our own approach is different than the ones presented above. The intelligent agent does not aid the students, but the employees of the IDS. Therefore, it is still an intelligent educational system, but students are no longer the main beneficiary. The current version of our implementation is a single agent system.

We implemented DTAP before designing the intelligent educational agent called Navibot. In order not to mingle with the source code of the original system, we chose to implement a plugin which is independent of DTAP and whose integration in the existing system requires minimal modifications.

The purpose of the plugin is to learn the sequence of actions executed by an employee of the IDS while managing a PhD thesis. Based on the acquired information, Navibot will make suggestions regarding the next action that should be executed considering the current state of the system. For instance, after inserting a thesis, documents a, b and c must be generated. Then, the information about the thesis is updated. In the end, documents d, e, f and g

can be generated as well. Keeping track of all these steps is not a big problem, but after doing this separately for every thesis every day, one may forget about generating a particular document or about updating a certain field of a thesis from time to time. It is in the nature of humans to commit errors, but preventing errors is always better than having to correct them. Hence, the purpose of Navibot is to help the users of DTAP manage the PhD theses without committing errors by pointing them the next step in the process of managing a thesis.

From the point of view of the user, the plugin is simple to use. The available use cases are the following:

- Display the window containing the hint.
- Hide the window containing the hint.
- Follow the action suggested by Navibot.

The plugin learns the sequence of actions dynamically while DTAP is being used. The benefit of this fact is that no supervisor is required to offer training data to the plugin in order for it to learn. Secondly, the plugin will automatically update its information at runtime when the sequence of actions changes.

As stated before, we used a RL algorithm for implementing the plugin. Navibot learns the optimal policy, the correct sequence of actions, from the feedback received for its suggestions.

The first step in designing such a plugin is to establish the set of states. In our case, the number of states used by the plugin is small, because only the relevant features of DTAP were considered. This decision was taken having in mind that this is a web application whose performance is influenced by the connectivity of the user to the Internet. Therefore, it was very important to keep the number of computations done by the plugin at a minimum. Otherwise, it would have affected the performance of DTAP and the plugin would no longer have been useful.

The set of states contains 10 elements, namely *Înregistrare teză (0)*, *Editare teză (1)*, *Vizualizare teze (2)*, *Generare Adresă minister (3)*, *Generare Anunţ susţinere (4)*, *Generare Decizia rectorului (5)*, *Generare listă documente (6)*, *Generare Proces verbal (7)*, *Generare Referat final (8)* and *Generare Referat preliminar (9)* where the string represents the name of the state and the number enclosed in parentheses its identifier. We associated to each state a unique identifier in order to use a simpler notation when referring to them.

The set of actions represents all the possible pages that a user of DTAP can visit. These actions will be displayed as links on the web pages. Users can access any of the above mentioned states. Therefore, the set of actions contains elements of the form "Go to state: s_i", where i represents the identifier of the

state. For instance, one of the actions in the set is "Go to state: s_1" which is equivalent with "Go to page: Editare teză".

Using the set of states and the set of actions we will define the set of transitions. There are no restrictions related to the navigation between the considered pages. Therefore, we can access any page from any other page of DTAP. Thus, the set of transitions contains all the pairs (s_i, s_j) where s_i, s_j belong to the set of states. An example of a transition would be (s_3, s_4) which can be expressed in natural language as follows. Starting from page "Generare Adresă minister" (s_3) go to page "Generare Anunţ susţinere" (s_4).

The learning process of the agent included in the plugin is driven forward by the received rewards. For each given suggestion which was wrong the agent receives a negative feedback. Otherwise, it receives a positive feedback. Therefore, the feedback, known in the literature as reward, can take either one of the following values:

$$r_k = \begin{cases} -0.5, \text{suggestion was wrong} \\ +1.0, \text{suggestion was correct} \end{cases}$$

Simply stated, the agent receives a return with the value "-0.5" for wrong hints and "+1.0" for correct ones. All the received rewards are cumulated in the return value "R" which is computed as follows:

$$R_k = \sum_{i=0}^{k} \gamma^{k-i} r_i$$

However, the updates are done at every step in order to reduce the space complexity of the plugin. Thus, the return value is updated at every step using the recurrence formula:

$$R_k = r_k + \gamma R_{k-1}$$

Experiments have shown that the plugin learns quicker when the value of γ is 0.9.

As the number of states is small our decision was to use the state-action value function. Therefore, the return value for each state-action pair (s, a) is stored. This information is stored in a matrix of dimensions N x N called *actions matrix*, where N represents the number of states. At the beginning of the algorithm every element of the matrix is initialized with the value 1, thus obtaining a matrix of 10 by 10 having all elements = 1.

In order to read the return of choosing the action starting from state 2 and which goes into state 4, the value of the cell with coordinates (3, 5) is read. Generally, in order to check which is the return of choosing the action starting from state i and which goes into state j, one must read the value of the cell

with the coordinates *(i+1, j+1)*. One of the benefits of this implementation is the access to the return values in $O(1)$.

All the concepts presented above are implemented in the Navibot plugin. The class diagram of the plugin is presented in figure 1. In order to be able to reuse this structure for all plugins using a RL algorithm, the ReinforcementLearning, State and Action classes are abstract ones.

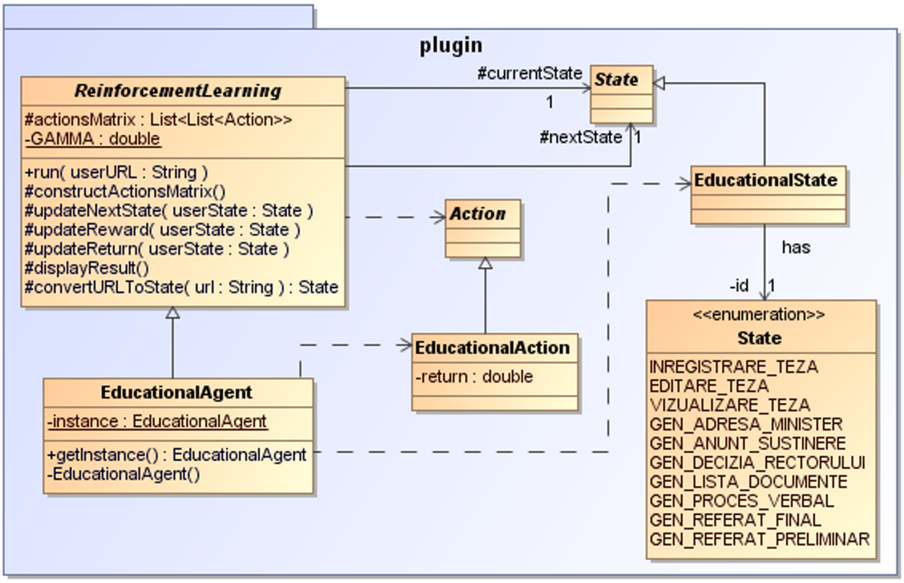

FIGURE 1. Navibot class diagram

The only public method defined in the ReinforcementLearning class is the method "run". This method describes the general behavior of a RL algorithm independent of the representation of the State and Action classes. The classes which are used for actually running the plugin are inheriting the abstract classes and overriding all abstract methods. In our case, these classes are EducationalAgent, EducationalState and EducationalAction.

The general public method "run" from the abstract class ReinforcementLearning is presented in the following using the Pseudocode notation. This method is called every time the user requests a new page. However, in order not to delay the delivery of the requested page to the user, the run method is called in an asynchronous manner using AJAX. As soon as the execution of the method "run" ends, the results are returned and the hint is displayed.

Algorithm 1 Public method *run*

```
1: get the current state of the application
2: if actionsMatrix is initialized then
3:    get value of reward
4:    update corresponding return
5:    choose the next state for the hint considering the maximum return
6: else
7:    construct actionsMatrix
8:    initialize next state
9: end if
10: display next state as hint
```

An execution of the method "run" will be described in the following. The description is given from an algorithmic point of view which means we will refer to states and not to pages.

Let us assume that the user wants to visit state 1. Therefore, the current state will be set to 1 (line 1).

If the action matrix is initialized, then the value of the reward is computed (line 3). The reward is "-0.5" if the previous suggested state is different from the current state or "+1.0" if they are equal. The return value corresponding to the previous suggestion is updated (line 4) using the value of the reward. Afterwards, the next state has to be determined (line 5). This is done by accessing the row corresponding to state 1 from the actions matrix. According to the explanations given above, this is row 2. Each column from the matrix corresponds to one state and has 1 value in the selected row. The algorithm finds the column containing the maximum value from the row. The state corresponding to this column is chosen as the next state.

On the other hand, if the action matrix is uninitialized, then it is constructed (line 7) and the value of all the cells is set to 1. The next state is initialized with a *NULL* value (line 8).

In the end, the hint is displayed (line 10). If the next state is different from *NULL*, then this state is recommended to the user. Otherwise, the hint will display a message which states that the plugin is currently being initialized.

Next, we will present some results obtained using the Navibot plugin. In figure 2 we describe the number of requests needed to learn sequences of actions of variable length. We mention that the number of requests depends also on which are the considered states.

We considered the number of requests as a unit of measure, because this is the most appropriate quantifier when working with web applications. We

also provide an estimate related to the time needed to learn the optimal policy assuming that a user makes an average of 6 requests/minute.

Our estimate is that the maximum time needed for learning a sequence, independent of its length, is less than 20 minutes. However, the sequence of actions changes only in exceptional cases, which means that once learned, the optimal policy will rarely change.

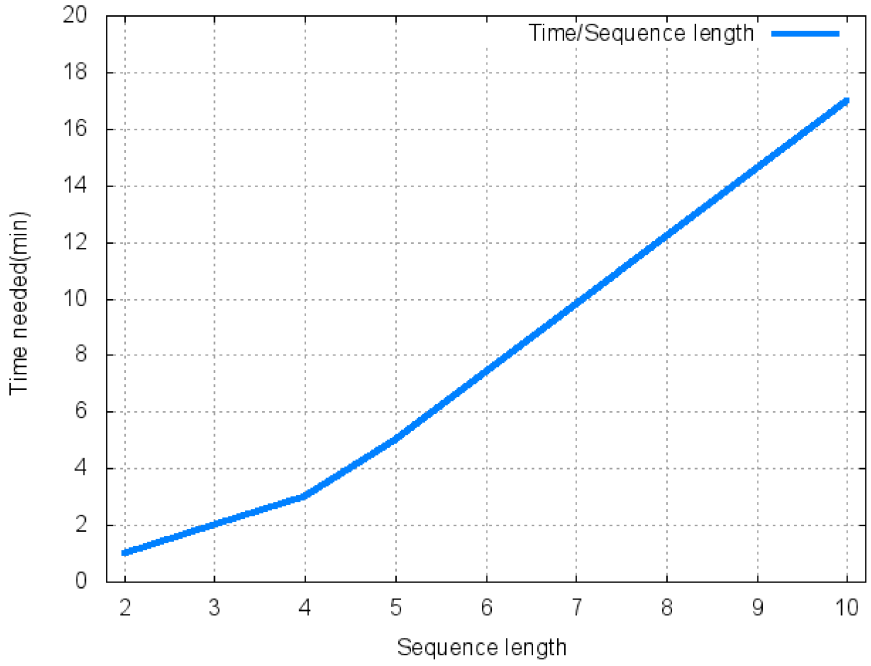

FIGURE 2. Navibot learning time

The schedule of the employees of the IDS assumes working 8 hours/day. Furthermore, the probability of committing errors at the beginning of the day is almost nonexistent, but it rises as the number of hours spent at work increases. Therefore, towards the end of the day when users need the help of Navibot most, the optimal policy will be learnt and accurate hints will be provided.

5. Conclusions and future work

We would like to emphasize the benefits of implementing the intelligent knowledge management system DTAP.

First of all, DTAP is an educational intelligent software system whose end users are the people working in administration and not the students or teachers. Its behaviour is changing dynamically based on its interaction with the users. The accuracy of the suggestions offered by the system increases throughout the learning process leading to good suggestions in less than 20 minutes. This guarantees that the number of encountered errors will be reduced after a relative short period of time.

The intelligent educational agent Navibot helps the users keep track of their progress when managing a thesis. Moreover, it prevents them from forgetting to make necessary updates or generate all the documents for a thesis.

A future version of Navibot will replace the single-agent system with a multi-agent equivalent. In order to achieve this, the present version of the plugin needs to be modified, such that all interactions of the users with the system are taken into consideration during the learning process. Results of an agent need to be persisted as a shared resource which can be accessed by all the other agents in the system. Therefore, we could no longer use a matrix for storing the rewards, but use a database instead.

We consider that our system should represent only the initial brick of a larger construction. The same principles which were incorporated into DTAP can be used for maintaining Bachelor and Masters theses. Moreover, the design of the application is generic such that only small adjustments are required for adapting it to the requirements of other universities. Therefore, DTAP could be used for managing the PhD theses in all the universities obeying the rules of the Romanian educational legislation.

References

[1] M.S. Abdullah, I. Benest, A. Evans, and C. Kimble. Knowledge Modelling Techniques For Developing Knowledge Management Systems. In *Proceedings of the 3rd European Conference on Knowledge Management*, pages 15–25, Dublin, Ireland, 2002.

[2] M. Alavi and D.E. Leidner. Knowledge Management and Knowledge Management Systems: Conceptual foundations and research issues. *MIS Quarterly*, 25:107–136, 2001.

[3] B.R. Barricelli, M. Padula, and P.L. Scala. Personalized web browsing experience. In *Proceedings of the 20th ACM conference on Hypertext and hypermedia*, HT '09, pages 345–346, New York, NY, USA, 2009. ACM.

[4] M. Chi, K. VanLehn, and D. Litman. Do micro-level tutorial decisions matter: Applying reinforcement learning to induce pedagogical tutorial tactics. 2010.

[5] M. Chi, K. Vanlehn, D. Litman, and P. Jordan. Empirically evaluating the application of reinforcement learning to the induction of effective and adaptive pedagogical strategies. *User Modeling and User-Adapted Interaction*, 21(1-2):137–180, April 2011.

[6] A.R. Gavin. *Problem Solving With Reinforcement Learning*. 1995.

[7] A. Iglesias, P. Martnez, R. Aler, and F. Fernandez. Learning teaching strategies in an adaptive and intelligent educational system through reinforcement learning. *Applied Intelligence*, 31:89–106, 2009. 10.1007/s10489-008-0115-1.

[8] T.M. Mitchell. *Machine Learning*. McGraw-Hill, New York, 2 edition, 1997.
[9] O. Pârvu. *Intelligent web application for managing the PhD theses in the Babeş-Bolyai University*. Cluj-Napoca, Romania, 2012.
[10] O. Pârvu and S. Dragoş. Interactive and intelligent doctorate theses administration platform. In *Zilele Academice Clujene (ZAC)*, 2012.
[11] M.J. Rosenberg. *E-learning : Strategies for Delivering Knowledge in the Digital Age*. McGraw-Hill Professional, New York, 2001.
[12] D. Stenmark. Information vs. knowledge the role of intranets in knowledge management. In *Proceedings of the XXXV Annual Hawaii International Conference on System Sciences (HICSS-02), 7-10 January*. IEEE, 2002. Exteneded version available from http://w3.informatik.gu.se/ dixi/.
[13] R.S. Sutton and A.G. Barto. *Reinforcement Learning: An Introduction (Adaptive Computation and Machine Learning)*. The MIT Press, March 1998.

Babeş-Bolyai University, Faculty of Mathematics and Computer Science, 1 M. Kogălniceanu St., 400084 Cluj-Napoca, Romania
E-mail address: ovidiu.parvu@gmail.com, sanda.dragos@ubbcluj.ro