International Journal of Research in Engineering, Science and Management
Volume-2, Issue-5, May-2019
www.ijresm.com | ISSN (Online): 2581-5792

Javashoot – A Web Enabled Intelligent Tutorial System that Teaches Java

Masoom Agrawal[1], Anjali Tripathi[2], Preeti Tuli[3]
[1,2]*Student, Department of Computer Science and Engineering, Shri Shankaracharya Institute of Professional Management and Technology, Raipur, India*
[3]*Assistant Professor, Department of Computer Science and Engineering, Shri Shankaracharya Institute of Professional Management and Technology, Raipur, India*

***Abstract*: The paper describes the design and development of JAVASHOOT, an Intelligent Tutoring Systems (ITS) that teaches java to university students. JAVASHOOT is a Web-enabled system, and we discuss its architecture and techniques used to deal with multiple students. We also discuss the plans for the evaluation of the system and future work.**

***Keywords*: Enter key words or phrases in alphabetical order, separated by commas.**

1. Introduction

Web-enabled educational systems are becoming the dominant type of systems available to students. Web based systems offer several advantages in comparison to standalone systems. They minimize the problems of distributing software to users and hardware/software compatibility. New releases of systems are immediately available to everyone. More importantly, students are not constrained to use specific machines in their schools, and can access Web-enabled tutors from any location and at any time. The time/location independence is of enormous value for learning environments, as flexibility and accessibility are extremely important for learning.

The Intelligent Computer Tutoring Group (ICTG) has been involved with developing intelligent tutoring systems for a number of years. Our system aims to provide immediate and customized instruction or feedback to learners, usually without requiring intervention from a human factor. It has the common goal of enabling learning in a meaningful and effective manner by using a variety of computing technologies. There is a close relationship between intelligent tutoring, cognitive learning theories and design; and there is ongoing research to improve the effectiveness of ITS. Countless domestic and global IT companies, multinationals and other IT firms are constantly looking for people with java skills. There are tons of good books on the internet, but you won't become a good programmer by reading books. To become a programmer you need to write a lot of code. JAVASHOOT. Intelligent Tutoring Systems designs differ significantly from their historical computer-driven predecessors. Rather than the one-size-fits-all strategy of delivering content to a passive learner in those designs, JAVASHOOT is able to customize the learning experience the student receives based on factors such as pre-existing knowledge, learning style, and the student's progress through the content material. In spite of the lack of visibility of its systems in the real world outside the rarified air of university research labs, there is a modest amount of research suggesting that intelligent tutoring systems can achieve remarkable increases in student learning over traditional classroom instruction.

2. Learning Java in Javashoot

A typical ITS will contain a number of conceptual components, or models, that interact with one another. The content model contains a web-like mapping of the content to be learned, defining the prerequisites and dependencies between the content elements. The student model is unique to each learner and works in parallel with the content model to record what the student does, and does not yet understand. Finally, there is a method of delivering the instruction to the learner, known as the pedagogical model.

Java is one of the most popular programming languages used to create Web applications and platforms. It was designed for flexibility, allowing developers to write code that would run on any machine, regardless of architecture or platform. In thinking about ITS, it is hard to envision a potentially more effective system for instruction. Such systems contain a semantically connected conceptualization of the content to be taught, a way of knowing what the learner does and doesn't understand, and a delivery method that adapts that instruction accordingly. It would appear that the early systems were not executed well enough to become mainstream; but they should, nonetheless, provide a rich foundation for future teaching machines to draw lessons from, as these systems begin to use the computer's power for more than simply delivering instruction. So in order to make the students much more adroit in java, we have created a platform where a candidate can come with a zero or little knowledge and expand his/her skills as demanded by various industries.

A login window will appear. To create a new profile, enter a name and password and click register, followed by log in.

-If you do not wish to create a profile, enter "user" and "password" in the respective fields for a default account.
-The slider will allow you to navigate the application.
-Example programming assignment:
For assignment 1 (Introduction) write in text area:
public static void main(String[] args){
System.out.println("Hello World!");}
and click submit or run.
-The system will write it to a class file, compile it, run it, and read the outputstream and match it to a key.
Additional Features for final build:
-The main user interface has had a massive overhaul
-The companion now speaks to the user via the text field on the left
-Companion speech is triggered by time. It follows the decorator pattern
-The motivational companion will recommend a break every 10 minutes.
-After 30 minutes, the motivational companion becomes the troll companion and taunts the user. (Can be adjusted or deleted)
-User may toggle guided companion. This follows the decorator pattern.
-When guided companion enabled, it will provide the user hints when the user clicks the next button on quizzes.
-If answers are correct, it will indicate so in the companion text panel.
-The control center now accounts for elapsed time. If you take longer than 5 minutes on a quiz or assignment, it will affect your standing.
-the control center is fully functional/will alter standing and face based on performance
-A total of 6 quizzes are now present and they save data to the user profile properly now.
-The content slides now contain buttons that will open helpful websites in a frame.
Flash cards have been added to assist in studying.
The user may view his/her progress via a panel containing progress bars.
If all six quizzes and all six assignments have been completed, and the student is in good standing (happy face), the next time the progress panel is accessed, it will display a certificate of completion in a frame.
Universe and program clock follow the observer pattern
Control Center, Companion Message Panel and Program Clock follow the singleton pattern.

The sequence of the steps is fixed: the student will only see a Web page corresponding to the current task. However, the student may ask for a new problem at any time during problem solving. In addition to that, he/she may review the history of the current session, or examine a global view of the student model. When the student submits the solution to the current step, the system analyses it and offers feedback. The first submission receives only a general feedback, specifying whether the solution is correct or not. If there are errors in the solution, the incorrect parts of the solution are shown in red. On the second submission, JAVASHOOT provides a general description of the error, specifying what general domain principles have been violated. On the next submission, the system provides a more detailed message, by providing a hint as to how the student should change the solution. The correct solution is only available on student's request.

Web-enabled systems use cookies or IP numbers to identify the student who made a request. Those two approaches were not suitable in our case. It was not possible to use the IP number, as several students might be using the same machine. We did not want to use cookies for identification purposes because cookies reside on a specific machine and would prevent the student from using the system from different machines. Instead, we identify students by their login name, which is embedded in a hidden tag of HTML forms and sent back to the server. If a student accesses a page by following a link instead of accessing it through a form, then user name is appended to the end of the URL. It is also necessary to store student-specific data separately from data about other students. All processing is carried out within a single address space, and therefore there must be a uniform mechanism for identifying students and associating requests to corresponding student models. In order to achieve this, we use a hash table that maps the string representing a student name to their student object, which contains all details pertaining to the student. Each action a student performs in the interface is first sent to the session manager, as it has to link it to the appropriate session and store it in the student's log. Then, the action is sent to the pedagogical module, which decides how to respond to it. If the submitted action is a solution to the current step, the pedagogical module sends it to the student modeller, which diagnoses the solution, updates the student model, and sends the result of the diagnosis back to the pedagogical module. The pedagogical module then generates feedback. If the student has requested a new problem, the pedagogical module consults the student model in order to identify the knowledge elements the student has problems with, and selects one of the predefined problems that feature identified misconceptions.

There are two feedback messages in the constraint, which are given to the student if his/her solution is incorrect. The first message is shorter, and tells the (5 (and (equalp (current-task sol) 'closure) (not (null (attribute-set sol))) (bind-all ?a (attribute-set sol) bindings)) (member ?a (closure sol) :test 'equalp)

"Each attribute that is an element of the set of attributes we want to compute the closure of must appear in the closure."

"Remember the reflexivity rule? Each attributes determines itself (A -> A).

The general form of the reflexivity rule is:
If X is a superset of Y, or X=Y, then X -> Y"
(?a "attribute-set"))

If the student still cannot correct the solution after this

message, JAVASHOOT will present the second message, which explains the underlying domain principle that has been violated (in this case, it is the reflexivity rule). The final element of the constraint specifies the part of the solution that is incorrect (in this case, that is the attribute to which variable a is bound). This binding is used for highlighting the error.

3. The architecture of JAVASHOOT

Figure 2 illustrates the architecture of JAVASHOOT. As can be seen, JAVASHOOT is based on a centralized architecture, as many other existing Web-enabled ITSs (e.g. ELM-ART [3], AST [11] and SQLT-Web [7]). Centralized tutors perform all tutoring function on the server side, where all student models are also kept. Distributed systems (e.g. ADELE [5], AlgeBrain [2] or Belvedere [13]) also keep the student model on the central server, but some of the tutoring functions are performed on the client. JAVASHOOT is developed in Allegro Common Lisp (ACL) [1] and uses the Allegro Serve Web server, which is an extensible server provided with ACL. At the beginning of interaction, a student is required to enter his/her name, which is necessary in order to establish a session. The session manager requires the student modeler to retrieve the model for the student, if there is one, or to create a new model for a student who interacts

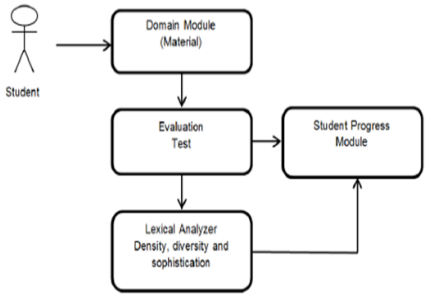

Fig. 1. Architecture of Javashoot

Fig. 1. A screenshot from JAVASHOOT

With the system for the first time. A Web-based tutor must be able to associate each request to the appropriate student model. Some Web-enabled systems use cookies or IP numbers to identify the student who made a request. Those two approaches were not suitable in our case. It was not possible to use the IP number, as several students might be using the same machine. We did not want to use cookies for identification purposes because cookies reside on a specific machine and would prevent the student from using the system from different machines. Instead, we identify students by their login name, which is embedded in a hidden tag of HTML forms and sent back to the server. If a student accesses a page by following a link instead of accessing it through a form, then user name is appended to the end of the URL.

4. Conclusions and future work

This paper presented the architecture and underlying philosophy of JAVASHOOT, a Web-enabled ITS for teaching database java. JAVASHOOT uses Constraint-based modelling to model domain knowledge and the knowledge of its students. However, unlike the previous tutors we developed, JAVASHOOT is the first constraint based tutor that teaches a procedural task. We have experienced no problems specifying constraints for such a task. The system contains a problem solver, capable of solving java problems. The knowledge base contains 53 constraints that check the syntax and semantics of students' solutions, enabling it to analyze all students' submissions. To analyze the semantics of solutions, JAVASHOOT compares the student's solution to the ideal solution produced by the problem solver. The number of constraints is likely to be higher, as we are currently working on the decomposition task. JAVASHOOT is a Web-enabled system, with a centralized architecture. Student models are kept on the server, and all tutoring functions are also executed on the server. The amount of information that needs to be transferred from the browser to the server is not large, and we believe that such architecture is appropriate. JAVASHOOT is developed in AllegroServe, an extensible Web server that allows the components of the system to be developed in Lisp. A special component of the system called the session manager ensures that a student's actions are associated with her/his student model, thus enabling the system to be used by multiple students simultaneously. We plan to evaluate JAVASHOOT in a real classroom in September 2002 at the University of Canterbury. The system will be used in an introductory database course, which has more than 170 enrolled students. We plan to compare the students' performance on a pre-test to their performance on a post-test, after using JAVASHOOT. Information about all sessions will be recorded in logs, and we will analyze how students learn constraints, and also evaluate other types of support the system offers, such as the open student model and support for self-explanation.

Acknowledgement

The work presented here was supported by the Computer Science Department, University of Canterbury. We thank Li Chen for developing the interface.

References

[1] Allegro Common Lisp, Franz Inc, 1998.

[2] S. Alpert, M. Singley, P. Fairweather, Deploying Intelligent Tutors on the Web: An Architecture and an Example. Int. J. Artificial Intelligence in Education, 10, 1999, 183-197.
[3] P. Brusilovsky, E. Schwarz, G. Weber, ELM-ART: An Intelligent Tutoring System on Wolrd Wide Web. In C. Frasson, G. Gauthier, A. Lesgold (eds), Proc. 3rd Int. Conf. On Intelligent Tutoring Systems (ITS'96), Springer, LCNS 1086, 1996, 261-269.
[4] R. Elmasri, S.B. Navathe, Fundamentals of database systems. Benjamin/Cummings, Redwood, 1994.
[5] W.L. Johnson, E. Shaw, R. Ganeshan, Pedagogical Agents on the Web. Proc. ITS'98 Workshop on Intelligent Educational Systems on the Web, 1998.
[6] M. Mayo, A. Mitrovic, Optimising Its behaviour with Bayesian Networks and Decision Theory'. International Journal on Artificial Intelligence in Education, 12(2), 2001, 124-153.
[7] A. Mitrovic, K. Hausler, Porting SQL-Tutor to the Web. Proc. ITS'2000 workshop on Adaptive and Intelligent Web-based Education Systems, 2000, 37-44.
[8] A. Mitrovic, B. Martin, M. Mayo, Using Evaluation to Shape ITS Design: Results and Experiences with SQLTutor. User Modeling and User-Adapted Interaction, 12(2-3), 2002, 243-279.
[9] A. Mitrovic, S. Ohlsson, Evaluation of a constraint-based tutor for a database language, Int. J. Artificial Intelligence in Education, 10(3-4), 1999, 238-256.
[10] S. Ohlsson, Constraint-based Student Modeling. In Student Modeling: The Key to Individualized Knowledge--based Instruction. Berlin: Springer-Verlag, 1994, 167-189.