Journal of Universal Computer Science, vol. 15, no. 10 (2009), 2078-2108
submitted: 18/2/09, accepted: 17/5/09, appeared: 28/5/09 © J.UCS

An Agent for Web-based Structured Hypermedia Algorithm Explanation System

Elhadi M. Shakshuki
(Jodrey School of Computer Science, Acadia University, Wolfville, Canada
Elhadi.Shakshuki@acadiau.ca)

Richard Halliday
(Jodrey School of Computer Science, Acadia University, Wolfville, Canada
040802h@acadiau.ca)

Abstract: Studying and understanding algorithms is important for all computer scientists. Over two decades of research has been devoted to improving algorithm visualization and algorithm explanation techniques. Knowledge gained from these practices allows us to design and implement logically correct programs with considerations to runtime and memory constraints. For many students, learning algorithms in a traditional manner (i.e. using text-books) is challenging. We have developed an alternative approach to teaching algorithms called the Structured Hypermedia Algorithm Explanation (SHALEX) system, which uses hypermedia and represents algorithms as an abstract tree structure. Although SHALEX is a fully functioning teaching tool, currently it does not provide a way of receiving feedback on student's progress. To address this problem, this paper extends SHALEX with intelligent agent to monitor student progress, to provide the student with hints where necessary and to record the results of student interaction, all of which provide a means of quantifying the level of understanding the student has achieved. The system is implemented as a web-based application using the client-server architecture. This allows students to learn algorithms through both distance education and in the classroom setting.

Key Words: Algorithm Explanation, Visualization, Hypermedia, HCI, Trees, Agents
Categories: H.5.1, H.5.2, H.5.4, I.2.4

1 Introduction

Nowadays, software is created using various development techniques and programming languages. Programmers can draw on a wealth of pre-designed algorithms to fit many programming tasks. The programmer selects the most appropriate algorithm and implements it. Computer Science, by nature, relies on an understanding of programming languages to implement ideas put forth. Just knowing a programming language is not enough for implementation as students must be aware of efficiency. Often, a program may be logically correct but its run time may be on the order of many years. The abstract concept central to an understanding of program behaviour, in terms of temporal and special requirements, is that of algorithms. Unfortunately, learning algorithms can be a difficult task for many students. a large amount of research has been conducted to understand how to effectively train people to use algorithms. One specific field that has received a lot of attention deals with

teaching and explaining algorithms (Braune, 00; Hübscher-Y., 03; Müldner, 08; Naps, 03).

A commonly used method for explaining algorithms is visualization (Hübscher-Y., 03; Naps, 03). Visualization uses animated or static visual content to aid in the understanding of an algorithm or problem. While this caters to visual learners, visualization is at best ineffective at teaching algorithms and at worst detrimental (Dijkstra, 89). This is because visualization techniques tend to focus on a single level of abstraction and frequently ignore aspects of algorithms such as invariants, i.e. properties that do not change during the execution. Furthermore, a relatively small number of techniques are typically used to deliver content, ignoring individual learning styles.

An Algorithm Explanation (AE) contributes to algorithm understanding by presenting an abstraction of the algorithm along with invariants, abstract input data and algorithm structure. It differs from algorithm visualization because it provides a set of tools that promote understanding of an algorithm, visualization being just one such tool (Demmings, 07). The motive behind AE is to ensure understanding of algorithms and to provide insight into algorithms that visualisation systems do not necessarily provide.

It is our aim to provide assistance to students in learning algorithms, through interactive visual aids and tests, with a Structured Hypermedia Algorithm Explanation (SHALEX) system (Shakshuki, 07; Müldner, 08). SHALEX uses the tree data structure to represent algorithms, which allows critical, but independent parts of the algorithm to be represented as lessons and then studied in isolation. In order to measure a student's progress in learning, each lesson is timed against a value the instructor feels is an appropriate length of time to learn a key concept of the algorithm. After the student has finished all lessons and consequently, the algorithm, a test is taken which will verify that the student understands this information. As the student becomes more knowledgeable about algorithms, they will exhibit different levels of understanding and will want different things from the system. In other words, as the user adapts, so should SHALEX. To monitor the student progress, an Algorithm Explanation Teaching Agent (AETA) is proposed.

The AETA is active for the duration of the time a student is using the SHALEX system. One of the main objectives of the AETA is to react to certain situations and to offer help if it is needed. Each user is given a skill level based on his or her previous cumulative performance in understanding algorithms. The help offered is tailored to one of four different skill levels: beginner, intermediate, advanced or expert. In addition to evaluating the student's score on the test, the AETA is responsible for evaluating a student's total understanding by comparing the time taken for the student to finish the algorithm to some instructor defined value. The AETA has the freedom to change the skill level of the student based on these values to provide the help best suited to each student. In this way, the AETA can directly manipulate the environment and provide SHALEX with the ability to change with the student as they gain knowledge and confidence concerning algorithms.

2 Related Work

Computer-Based Training (CBT) systems provide linear training capabilities for the algorithm visualization and explanation domains (Baker, 04; CIMEL, 06). CBT systems offer simple hyperlinks to visualization or help content, as an augmentation to algorithm lessons, and so they give the advantage of offering wider, more varied content. However, CBT systems lack adaptability which can be provided by Artificial Intelligence techniques, specifically Intelligent Tutoring Systems (ITS). ITSs have drawn on pedagogical knowledge and a student model to facilitate decision making (Shi, 00; Moritz, 05; Wei, 05). Here, the so-called pedagogical agents (PA) are responsible for evaluating the content of the student model and tutoring the student based on the specific errors the student makes and on the curriculum in general.

The work presented in (Moritz, 05; Wei, 05) detail a specific example of a tutoring system that makes use of a pedagogical agent. The Constructive, Collaborative, Inquiry-based Multimedia E-Learning, CIMEL (CIMEL, 06) tutors users who are programming in the Java programming language. In CIMEL, the PA draws on a student model and a curriculum model to tutor programmers based on their mistakes; while an expert evaluator evaluates student code and updates the student model. A student's progress is compared against a solution found within the curriculum model (Moritz, 05). If the evaluator finds inconsistencies between the solutions that the student provides and the expected solutions found in curriculum model. However, the PA does not use temporal context such as the length of time between student actions, to aid in providing assistance. It is stated that the pedagogical agent "determines the timing, style and content of the teaching agent's interventions" (Shi, 00). It reacts to changes in the student model and the curriculum model to ensure the student develops an understanding of the material.

One side effect of offering an online assistance to students is the possibility that students will request help as rapidly as possible. Student may build sufficient short-term knowledge to derive the correct answer, but not necessarily an understanding of the correct answer. This is referred to as "gaming the system" because of the rapid cause and effect feedback experienced, similar to many video games. Baker, et al. (Baker, 04) investigated this problem and defined gaming as being "…systematic and rapid incorrect answers or use of help, with several such actions taking place during the period of 20 seconds…"

Inactivity is another possible indicator of lack of comprehension, or inattention to the task. Baker et al. (Baker, 04) found a significant correlation between student inactivity during tutoring and that student's post-tutoring test results. Their study makes a distinction between inactivity measured by the student doing nothing at all, and task inactivity, where the student may be talking about a subject unrelated to the task (i.e. Talking Off-Task). This distinction exposed a "marginally significant" negative correlation between talking about an unrelated subject and post-tutoring comprehension tests.

With the advent of faster computers and Internet access, web applications (Rahayu, 03; Zaupa, 08) and software agents (Dudek, 08) are widely disseminated, and distance learning has become more available and more popular. Although distance learning is attractive in that the student does not have to leave home to attend classes, there are several drawbacks which prevent distance learning from providing

education on the same level as classroom-based teaching. One major drawback is how the course is delivered. If students receive only text-books and some assignments, then they might have been just as well off learning on their own and not paying the fee. This is especially problematic in the domain of teaching algorithms as often text-books written on algorithms are difficult to study without outside assistance.

Research presented on teaching algorithms suggests there is not just one correct teaching approach (Curcio, 98). The research done in (Müldner, 08) outlines the SHALEX system which is a graphical tool used to teach algorithms. To combat the sequential nature of text-books, SHALEX incorporates hypermedia. The nature of hypermedia allows the student to follow their own path through the algorithm lessons and enables two students to approach the same algorithm in two different ways.

The work in (Müldner, 06) recognises the usefulness of visualisations in helping students gain an understanding of algorithm principles. Visualisations offer a way for the student to relate non-tangible aspects of the algorithm by way of graphics and animations. SHALEX allows the author of the algorithms to include visualisations by way of a programming language designed especially for constructing them. Research conducted in (Grissom, 03) agrees that visualisations alone are not enough to adequately teach a student about the complexities of algorithms. With this in mind, algorithm visualisation is kept as an option in SHALEX system.

In addition to hypermedia, SHALEX further frees the student from sequential learning by representing algorithms in a tree structure. Moreover, the algorithm being studied is represented by the root node of a tree and the important, independent concepts, which make up this algorithm, are represented as the child nodes. The author of the algorithm can suggest the best approach for studying this algorithm; but ultimately, the students are free to study any node in any order.

The second issue with distance learning is the lack of a teacher with whom to interact. In a classroom setting, students have the ability, and are encouraged, to ask questions whenever they need assistance following a lesson. The research in (Kutay, 05) focuses on using an agent as a means of evaluating student work. In that paper, agents are designed to review documents created by students and provide feedback. This approach requires the student to finish before the agent can provide feedback. The agent proposed in this paper monitors the students in real time, but still incorporates the idea of offering feedback from a computer-based teacher, or expert.

I-Help (Vassileva, 03) is a tool which grants its users access to electronic help and peer tutors by matching users who need help with users who can offer the best answers. Each user has many agents, each of which maintains their own model of the user. It is likely that no two agent's models are alike. The work in (Fok, 06) stresses the importance of agent adaptability. Our proposed agent incorporates these ideas in an attempt to change with the user and provide appropriate feedback when necessary.

In our previous work (Shakshuki, 04; Demmings, 07), the concept of soft-times and hard-times to measure students progress is introduced. Each lesson has a soft-time and the sum of these soft-times is the hard-time. The soft-times are variable so that students can take a longer time to finish a lesson, if need be. The hard-times are invariable, so the sum of all soft-times must always be equal to, or less than, the hard-time. In order to keep this relation true, the extra time required to finish a lesson is subtracted from the other remaining soft-times. The AETA proposed in this paper, which is an extension to the work presented in (Shakshuki, 08), uses the same method

for measuring progress; but, the technique for penalising the student and reducing remaining soft-times takes into consideration the fact that as a student progresses through a set of lessons, he or she will be able to finish subsequent lessons more quickly. Therefore, the AETA still deducts extra time needed; but, in such a way as to penalise the later lessons more than the next lessons.

3 Problem Domain

SHALEX is an educational teaching system tool that utilizes hypermedia in order to offer students a richer experience in learning algorithms. In the case of SHALEX, hypermedia refers to embedded hyperlinks. The benefit of hyperlinks within a teaching environment is twofold. First, hypermedia does away with sequential learning. Without hyperlinks, the student is resigned to learn an algorithm in the order the author feels most appropriate. Adding hyperlinks allows greater freedom for the student to follow his or her own path to the goal of understanding. Second, hyperlinks allow for system integration. It is often easier to understand a topic if it is analogous to something which one is already comfortable. Hyperlinks make it possible to include similar algorithms, which exist on the system, in the description of the algorithm being studied. SHALEX incorporates several algorithm teaching techniques into one teaching method that gives choice to the student. These methods include visualizations, examples written in pseudo-code, a description in plain English including hyperlinks, and optional extra help which may include anything the author feels is beneficial to the student. There is a test associated with each algorithm that the student must complete when he or she is satisfied with his or her level of understanding.

SHALEX is designed to allow the addition and removal of algorithms within the system. As well, previously created algorithms can be updated and edited. These operations are accomplished through the author user type. This user is assigned to someone with knowledge of algorithms, their implementation, and the best ways to convey this information to students. Algorithms are stored in SHALEX using the tree data type with the algorithm itself as the root lesson, and the various critical components of the algorithm as the children lessons. Using this structure, it is possible to isolate important pieces of the algorithm for individual study. When the author creates a new algorithm, he or she first creates all of the lessons that will be used to compose the tree.

Each lesson in the algorithm tree is given a soft-time by the author. This soft-time dictates the length of time the author feels is necessary to gain a satisfactory understanding of the concepts presented by the lesson. We call this a soft-time because it is variable depending on the student's actions. The sum of the soft-times of the lessons in the tree, prior to the student starting the lesson, is called the hard-time for the algorithm. The hard-time is not variable and therefore it is used to determine if the student finished studying early or on time. Finishing an algorithm early or on time is defined by the student finishing all lessons before or right at the hard-time. Note that the soft-time for each individual lesson does not directly determine whether or not the student is early or on time. A student is considered late when the sum of the soft-times is greater than the expected sum of the soft-times. Also, lessons can have a

number of Abstract Data Types (ADTs) associated with them; these represent data structures vital to the understanding of the algorithm.

The author creates all of the lessons that make up the algorithm by providing a description, pseudo-code, and extra help to each. The tree is created by setting the algorithm in question as the root lesson, and arranging the remaining lessons as dictated by their relation to one another in the actual algorithm that is being represented. Once the algorithm is created, a strategy must be assigned. Algorithm strategies, in the domain of SHALEX, are defined by the approach taken to learn the algorithm. Since we are working with a tree structure, the three strategies which can be applied are: top-down, bottom-up, and mixed. Mixed is a combination of top-down and bottom-up. The top-down strategy presents the root lesson first which may be overwhelming to some students as the root lesson consists of a description of the entire algorithm. The bottom-up strategy begins with the leaf lessons; those parts of the algorithm which are less intimidating to beginner students, but may be detrimental to more advanced students who wish to first see the entire algorithm before they study the components. The mixed strategy combines both of these approaches.

From the above discussion of the strategies, it is easy to see the benefit of having a way to differentiate among students at different levels of understanding with regard to algorithms in general. SHALEX includes four different skill levels which can be assigned to students: beginner, intermediate, advanced, and expert. This allows the author to tailor the same algorithm to four separate groups of students by incorporating the idea of contextual help. The main idea of contextual help is the help a beginner level student requires should be more descriptive in nature and based around just the main concept of the algorithm or lesson, whereas the help an expert level student receives should be focused more on the implementation and code. SHALEX is implemented to allow the author to define one lesson containing the four levels of help, which is then presented to the student, as appropriate.

Studying the lessons that make up the algorithms is where the “real learning” takes place in SHALEX. As the student reviews the information in the lesson, he or she has the option of finishing the study, or merely stopping the study. If the student stops studying the lesson, he or she may resume at a later time from the point where he or she left off. If the student finishes the lesson, the student may not resume and it is assumed he or she understands the lesson completely. Once all of the lessons have been completed, the study of the algorithm is considered to be complete and the student is now required to answer the questions in the test to display the newly acquired knowledge. As the student completes the lessons, algorithms, and the tests, the information is recorded and the section of the SHALEX knowledge base allotted to that student is updated.

The objective of the AETA is to monitor the student’s progress through the SHALEX environment. As the student progresses, he or she manipulates the environment through interactions to the interface and also the knowledge base. These interactions are referred to as events. The firing of these events drives the AETA to make decisions based on the data at hand, which causes the AETA to interact either with the student directly, or with the SHALEX system. The two types of events processed by the AETA are action events and temporal events. Action events occur as a result of the student “physically” manipulating the environment through use of the graphical user interface (GUI) and consist of system actions and help actions.

Temporal events follow the expiration of the soft-time associated with the lesson being studied.

System actions are events the student triggers by interacting with the system through clicking the buttons in the GUI. For example, the student chooses an algorithm to begin studying and clicks on the "Study Selected Algorithm" button. This opens the dialog to choose the lessons available in the algorithm and also activates an event that tells the system the student has begun to study the algorithm. These interactions with GUIs will be discussed in more details in the communication section.

The AETA listens for all such events and uses them to determine the student's progress through the lessons and within the system. System actions are activated by:

- Starting and stopping algorithm study.
- Starting and stopping or finishing lesson study.
- Starting and stopping or completing a test.

The help action is a special action event that can be invoked by the AETA or the student. In all lesson dialogs, there is a button to request help. The user clicks this button to initiate the help request event, which may or may not display a helpful message. For instance, if the number of help requests becomes excessive over the course of the lesson, it may be an indication that the student is trying to memorize the help instead of gaining a full understanding of the concepts. In this case, the AETA will not ask the student if help is required, nor will help be provided if the student clicks the help button.

Temporal actions are triggered by the system timers, specifically, the process that monitors the soft-time when the student is studying a lesson. In order to guide the student to the correct solution, some assumptions are made about human habits. For instance, if the student remains inactive for long periods of time, it could be that he or she is having trouble with the material, or merely left the workstation unattended. Since we cannot detect which is the case, we prompt the user with the help action to let the user decide if help is required.

4 Environment

SHALEX is designed with a client-server architecture in mind. The SHALEX program and data are stored on a central server and each student will log into this server using a web browser which activates the SHALEX Java (Sun Microsystems) applet, as shown in Figure 1. The applet is the platform which contains the system GUI along with each student's unique AETA which runs inside the applet. The AETA has access to all input from the user to the system and will be able to communicate with the user through the use of dialog boxes. The AETA demonstrates learning by updating the system's knowledge base as a result of student actions within the system. It should be noted that the purpose of the AETA is to observe student interactions with the system and respond to the situations at hand based on its knowledge and reasoning techniques.

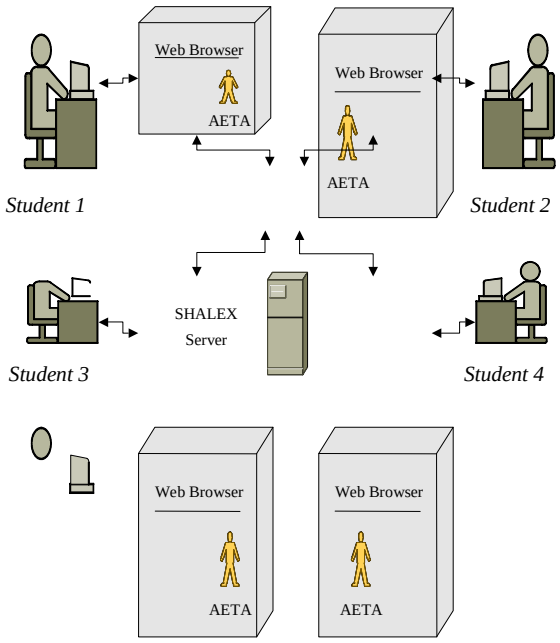

Figure 1: The SHALEX environment

The server holds all of the data in the system. This includes, user information (username, password, skill level, etc.), lesson information (descriptions, soft-times, help messages), and the AETA for each individual user on the system. An algorithm in SHALEX is stored as a collection of lessons. The set of algorithms *A* contains all algorithms in the system and is comprised of the set of all lessons. Formally, we consider the set of *n* lessons, $L = \{L_i \mid 1 \le i \le n\}$, which includes all of the lessons in the system. Together, these lessons make up all of the algorithms on the system. For any algorithm *a* in *A*, *a* is composed of *j* lessons. We can represent these *j* lessons as L_k where $1 \le k \le j$ and $1 \le j \le n$.

Each lesson has an associated soft-time which is defined as the length of time the author feels is appropriate to finish studying the lesson. These times are soft-times because they are variable depending on if the student is early or late in finishing the lesson. The sum of the lessons' soft-times makes up the hard-time for the algorithm. The hard-time is invariable and is always equal to the sum of the initial values of all soft-times. For example, assume an algorithm is made up of three lessons: Lesson 1, Lesson 2, and Lesson 3 with initial soft-times S1, S2, and S3 of 60, 60, and 180 seconds respectively, as shown in Figure 2. The hard-time, H_t, for this algorithm is equal to S1+S2+S3 = 300.

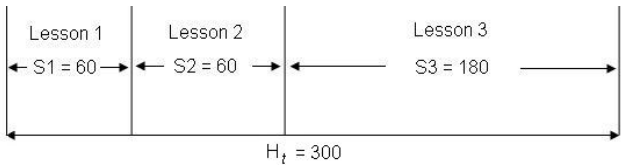

Figure 2: The initial soft-times of the three lessons

To continue our example, assume the student needs 40 extra seconds to gain an understanding of Lesson 1 and Lesson 2. This will cause the new soft-time of Lesson 1, S1', to be 100 seconds and similarly for Lesson 2. Since the hard-time of the algorithm is a constant, the soft-time of Lesson 3 must decrease by 80 seconds to accommodate S1' and S2'. The new hard-time diagram is shown in Figure 3 with the original intervals pictured as solid lines and the updated intervals as dotted lines. It is clear that H_t remains equal to 300 seconds.

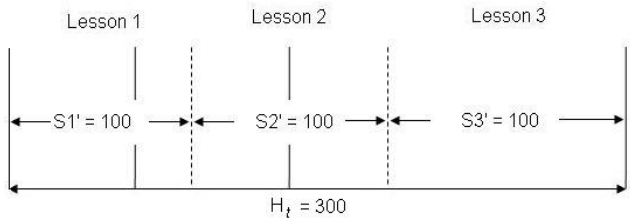

Figure 3: The updated soft-times of the three lessons

The AETA makes its decisions based on the student's data and the current state of the lessons. The lessons change their state based on relationships they have to one another and themselves. For example, in order for the AETA to properly offer help to the student, the AETA must be aware of the current elapsed time compared to the soft-time for the current lesson. These values are acquired through the lessons' relationships. A relationship in SHALEX is of the unary type which means these relationships take a single parameter. Some examples of relationships among the lessons on the system can be described as follows:

- Lesson_Started (x), $x \in L$: This is a unary relationship that returns true if lesson x has been initiated by the student.
- Algorithm_Finished (x), $x \in A$: This is a unary relationship that returns true if all n lessons that make up the algorithm have been finished. Formally, it returns true if Lesson_Finished (L_i) is true for all L_i, $1 \leq i \leq n$.

Elapsed_Time (x), $x \in L$: This is a unary relationship that returns the time accumulated from the moment the student starts studying the lesson.

5 Architecture

The AETA is an interface agent and its architecture is composed of a set of components, as shown in Figure 4. This architecture includes the problem solver, communication, learning, scheduler, and the knowledge components that interact with and update the student model and author model. The student model contains a set of dynamic characteristics, which includes everything the student has accomplished with the system concerning the completion of lessons, algorithms, and tests. The student model also stores a set of static characteristics such as the username, the user's real name, and the user's skill level. The skill level characteristic is considered static because it is updated much slower than other dynamic characteristics. The author model consists of the values against which the student's performance is measured; this includes hard-times and soft-times for the lessons and correct answers for the tests.

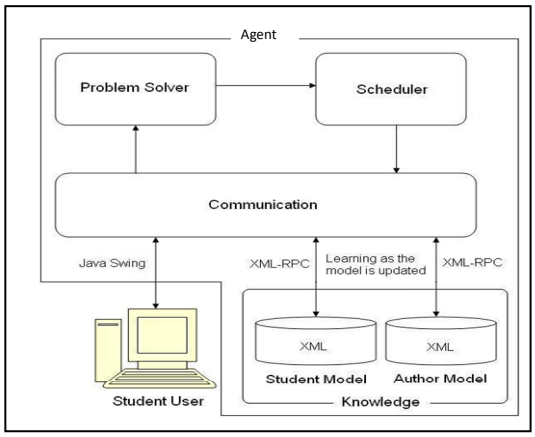

Figure 4: Agent architecture

5.1 Problem Solver

The problem solver component works like a function in that it takes in some input (i.e., problem), processes the input and delivers an output (i.e., solution). The problem is defined to be the student's complete understanding of everything represented in the author model component before the allotted time expires. The output, here, is the solution to this problem, which is made up of a schedule consisting of appropriate times for a student to finish the lessons that make up the algorithm in question. The problem solver uses the knowledge gathered on the student, based on his or her model, such as the skill level and the current state of the lesson including the soft-time and the elapsed time.

5.1.1 Problem Solver Implementation

The AETA is a rule-based agent; therefore, the problem solver is implemented as a set of conditional statements which are evaluated based on the student's current progress, the student's history, the data stored in the author model, and the values returned by the current timers. This makes it possible for the AETA to decide whether its next action should be to interact with the student, or to update the knowledge on the system.

The student interactions can be broken down further into two sub-categories: student inputs and student outputs. We shall define student inputs as anything requiring the student to make a decision and/or interact with the system via button clicks. We define student outputs as anything that informs the student but does not require anything from the student. Student inputs used by the AETA are Warn_Student, Show_Help, and_Ask_Student. Student outputs used by the agent are Display_System_Message and Display_Agent_Message. For example, the Warn_Student action is issued when the student spends more than the system-defined time studying a lesson. The student is warned first when the lesson timer decreases past a certain threshold of the original soft-time. A warning is issued stating that the student has spent that threshold of time studying and is expected to be at a certain stage of understanding. Another warning is issued when the student has used up all of the soft-time. The timer continues to decrease, and the time in excess of the soft-time is deducted from the soft-time of the remaining lessons. In both of these cases, the student is given the choice to either accept or deny help.

System interactions consist of those actions done by the agent which update the knowledge of the system, particularly the student model. The AETA interacts with the system via a set of actions, such as Start_Lesson, Stop_Lesson, Start_Algorithm, Start_Timer and Update_Student_Skill_Levelm. For example, Update_Soft_Time action is called by Finish_Lesson and is initiated when the student finishes a lesson after the proposed soft-time. The amount of time the student needed over and above the ideal soft-time is redistributed among all remaining lessons. The new soft-times are then saved to the student model. Another example, Update_Student_Skill_Level action transpires as a result of a decision by the AETA based on the number of algorithms completed, N, and the student's total performance value, P_T. The number of algorithms completed is an administrator-defined value that dictates the incremental number that must be reached before the AETA will perform the skill level evaluation. This evaluation results in the student's skill. Thus, it increases to the next level if the value of P_T is greater than or equal to $0.5 \times N$. Students cannot be demoted to lesser skill levels. For example, if the number of algorithms is set to 5, P_T will be evaluated to $0.5 \times N$ after the completion of 5, 10, 15, etc., algorithms. Formally, P_T is measured by three variables P_i, C and T and can be recursively defined by the following equation:

$$P_T = P_i + 0.15 \times C_{i+1} + 0.85 \times T_{i+1}$$
$$P_i = P_{i-1} + 0.15 \times C_i + 0.85 \times T_i$$
$$i \geq 1, P_0 = 0.$$

Where, C_i is given a value of 1 if the student finishes the i^{th} algorithm either early or on time. If the student finishes late, C_i is given the value 0. T_i is the score of the test which must be written after the i^{th} algorithm is completed. It can be seen here that if the student finishes an algorithm late, but gets 100% on the test, the student's total performance value, P_T, for that algorithm will be 0.85. Conversely, if the student finishes the algorithm on time, or early, and gets 0% on the test, the total performance value will be 0.15. This is designed so students, who rush through algorithms to finish early or on time, but get low test scores, are not given increased skill levels.

AETA as a rule-based agent, a set of well-defined actions are formed for AETA to follow. Formally, these consist of a set of pre-conditions and a set of post-conditions for each rule. For example, a Start_Lesson action has the pre-conditions $Lesson(x) = \ell$, $\overline{Lesson\_Started(\ell)}$, $\overline{Lesson\_Finished(\ell)}$, $Lesson\_Stopped(x)$ and post-conditions $Lesson(x) = \ell$, $Lesson\_Started(\ell)$, $\overline{Lesson\_Finished(\ell)}$, $\overline{Lesson\_Stopped(\ell)}$.

5.2 Knowledge

The knowledge of the AETA serves as the non-volatile memory of the agent, this means information is retained after the system is turned off or the application is exited. The agent's knowledge is dynamic and is updated through student interactions with the system. The knowledge is used by the problem solver to generate solutions which guide the student through the algorithm being studied. Knowledge is represented as two sets of data, the author model and the student model. The author model consists of information that reflects an ideal student's progress through the lessons. Actual student progress is measured against this model in order to fully evaluate the student. The author model is composed primarily of soft-time and hard-time measurements. The soft-times are entered by the author as the lessons are created while the hard-time for the algorithm is automatically generated as the sum of the soft-times. In addition to these temporal attributes, the author model also contains a set of correct answers to the test associated with each algorithm and a help listing for each lesson for every skill level. The author model is comprised of these four attributes:

- Soft-times: The time required for an ideal student to complete the lesson at hand.
- Hard-times: The sum of the soft-times in an algorithm.
- Test Answers: The correct answers to the tests are matched against answers given by the student.
- Help: Help is provided to the student by the AETA. Help is contextual so students with different skill levels will receive help more tailored to their level of understanding.

The student model contains information unique to each student user on the system. This information is matched against the ideal information in the author model in order to determine the student's progress within the SHALEX system. The student model records these three attributes:

- Skill Level: The skill level of the student is used to determine what help the student will receive by the AETA. There are four skill levels (beginner, intermediate, advanced, expert) which are assigned when the account is created. The skill level is updated by the AETA as the student completes lessons, algorithms, and tests. The skill level can also be edited by a system administrator.
- Time Remaining: As a student studies a lesson, a timer will count down from the lesson's original soft-time to zero. The time remaining is given by the lesson's soft-time minus the total elapsed time. If the time remaining reaches a value less than zero, this extra time is subtracted from remaining lessons.
- Completed Test Scores: A record of the percentage of questions answered correctly by the student, versus the number of questions.

The AETA uses both the author model and the student model to monitor the student's progress, offer help, and update the user's skill level. The goal of the AETA is to provide the student with an understanding of all the lessons contained in an algorithm within the hard-time specified by way of graphical interactions. The AETA uses the author model to determine the soft-times associated with these lessons. From the soft-times, hard-times, and the current elapsed times, the AETA can schedule student interactions and provide help when necessary.

5.2.1 Knowledge Implementation

Both the author model and student model are stored in XML format in an eXist (Open Source Native XML Database) database (eXist, 08). XML databases provide a platform-independent means for storing and accessing data that is key for our client-server architecture. The SHALEX system accesses the eXist database through XML-RPC, which allows for the adding, deleting, and editing of data with a Java user interface.

The student_model consists of two main XML documents which are unique to each student in the system. Figure 5 shows the first document, which manages the elapsed time for each lesson and is indexed by the ID number of each lesson. Lessons are represented by nodes tags in the XML file.

```
<?xml version="1.0" encoding="UTF-8"?>
<nodes index="6">
    <node id="4">
        <softTime>22</softTime>
    </node>
    <node id="5">
        <softTime>23</softTime>
    </node>
    <node id="6">
        <softTime>13</softTime>
    </node>
</nodes>
```

Figure 5: An XML representation of elapsed time

Figure 6 shows the second document, which is responsible for student characteristics.

```
<?xml version="1.0" encoding="UTF-8"?>
<users index="6">
    <user id="4">
        <username>student</username>
        <password>student</password>
        <role>Student</role>
        <fullName>Student One</fullName>
        <email/>
        <description/>
        <FontName/>
        <FontSize/>
        <skillLevel>Beginner</skillLevel>
        <memberOf>
            <group id="1"/>
        </memberOf>
        <completedNodeList>
        <node id="4" name="QuickSort" time="-17"/>
        <node id="5" name="Partition" time="45"/>
        <node id="6" name="Recursively Sort" time="50"/>
        </completedNodeList>
        <completedStrategyList>
            <strategy id="2" result="On Time"/>
        </completedStrategyList>
        <notCompletedTestList/>
        <completedTestList/>
    </user>
</users>
```

Figure 6: An XML representation of the student model

These characteristics are divided into two categories: static and dynamic. The static characteristics are attributes of the student which never change such as user name and the real name of the student. Dynamic characteristics change as the student progresses through the lessons. Skill level, number of algorithms completed, test scores, and the names of the completed algorithms and lessons are examples of dynamic characteristics.

The author_model is made up of an XML document that contains the lessons which make up each algorithm and is shown in Figure 7.

```
<?xml version="1.0" encoding="UTF-8"?>
<nodes index="3">
    <node id="2" name="Pivot" ref="1" softtime="60">
        <adts/>
        <description>The pivot is one of the main concepts of the
                     quick sort algorithm. A pivot is chosen from
                     one of the various techniques and then all
                     elements smaller than the pivot are moved to
                     one side and all elements larger than the
                     pivot are moved to the other side.
        </description>
        <code>function partition(array, left, right, pivotIndex) {
                pivotValue := array[pivotIndex]

                        // Move pivot to end
                swap(array[pivotIndex], array[right])
                storeIndex := left

                for i from left to right-1 {
                  if array[i] > pivotValue
                        swap(array[storeIndex], array[i])

                  storeIndex := storeIndex + 1
                }

                        // Move pivot to its final place
                swap(array[right], array[storeIndex])

                return storeIndex
            }
        </code>
        <visualization/>
    </node>
</nodes>
```

Figure 7: An XML representation of a lesson

These lessons are given a unique index which is passed along to the student_model XML files. The soft-times that indicate the ideal time to finish each lesson are stored in this document along with the help that can be offered to each skill level by the AETA.

5.3 Scheduler

The scheduler of the AETA is used to order the actions of the agent with respect to time. The appropriate actions come as a result of the problem solver component generating a correct solution. The problem solver uses the ideal times stored in the author model along with the current elapsed times stored in the student model and determines if the AETA should schedule a student action or a system action.

5.3.1 Scheduler Implementation

A schedule exists as an ordered triple which takes on the parameters of lesson ID, the soft-time associated with that lesson, and the time remaining (soft-time minus the elapsed time). When the lesson is loaded for the first time, the time remaining equals the soft-time. The timing device used to measure the elapsed time relies on the hardware clock of the student's computer. The timer is run as a separate thread to allow for student interaction to occur as the elapsed time increases. This makes it possible for the AETA to interact with the student while they click the mouse and read the lesson. For each second the timer counts, one second is removed from the time remaining. When the time remaining reaches some pre-defined threshold, either a student interaction or a system interaction is scheduled.

5.4 Communication

The communication component enables the AETA to interact with the student and the SHALEX server. This is the means for the AETA to provide the user with help and to update the knowledge base. The AETA interacts with the student through the use of dialog boxes. Help is also provided with a dialog box and it should be noted that the timer continues to increment the elapsed time while the help is visible, which attempts to deter the student from spending the entire lesson time reading the extra help. Continuing the timer is also a necessity, as it stops the student from being able to move the help dialog off the visible screen which would result in the student having unlimited time with the lesson. Since the student can also request help on his or her own, the AETA must monitor the amount of times per lesson the student is making these requests. If the number of requests exceeds a number defined by a system administrator, the AETA will step in and deny help to the user until the next lesson. If the student finishes a lesson after the time allotted by the author, the extra time taken is redistributed and subtracted from the time remaining in all of the other lessons that have not yet been completed. If the student finishes the algorithm after the specified hard-time, then the student is late. The AETA records one of the three possibilities (late, early, on time), for each algorithm, in the student model for determining the student's skill level.

5.4.1 Communication Implementation

The student interacts with the SHALEX system through a Graphical User Interface (GUI) implemented with Java Swing. After the student successfully logs in, he or she is presented with a panel, as shown in Figure 8, which gives the basic overview of the algorithms the student is to be learning.

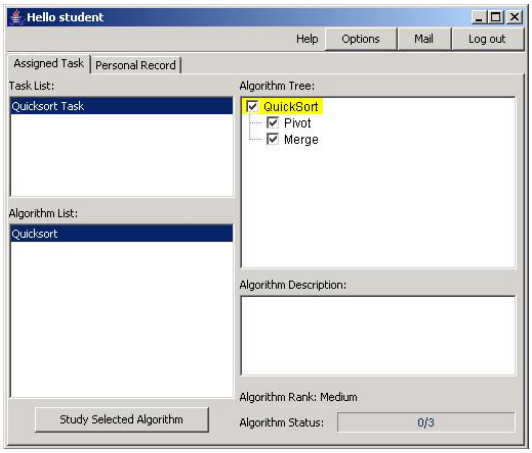

Figure 8: A dialog of tasks for the student

In this case, the user selected the "Quick Sort Task" and then the "QuickSort" algorithm, which displayed the tree representation of the quick sort algorithm, as shown in Figure 9.

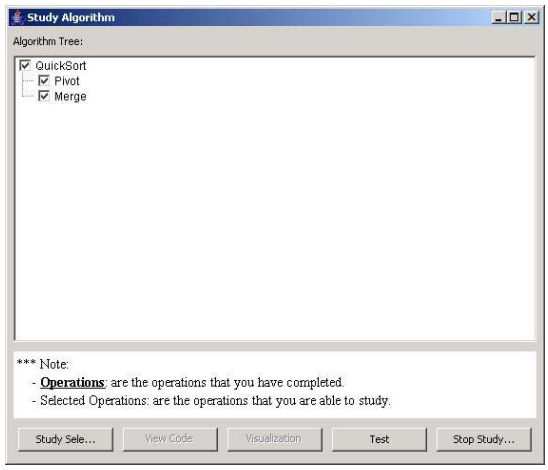

Figure 9: A dialog displaying the QuickSort tree

From here, the student is allowed to choose one of the lessons in this algorithm tree. These lessons are the lessons that make up each algorithm explanation. At this stage, the AETA has acquired the necessary information to seed the scheduler with the soft-times, and the lesson ID numbers from the author model. The check boxes represent lessons that have not been completed. The student chooses the root lesson of the algorithm tree which opens the main lesson dialog for the "quick sort" lesson, as shown in Figure 10.

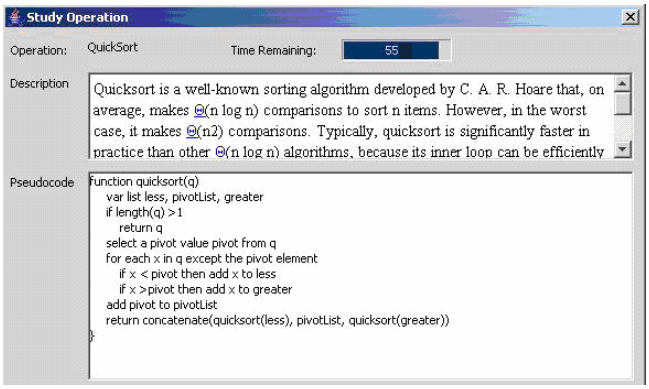

Figure 10: Main lesson dialog for QuickSort

As soon as the lesson dialog opens, the time remaining begins to decrease with every second of clock time as measured by the student's computer's internal hardware clock. This enables us to determine an accurate measure for when the student begins learning. Each lesson has one or more timing thresholds that will cause the AETA to alert the student of the time remaining and ask if help is required. In this example, there are two such thresholds. The first threshold is reached when the time

remaining= (0.5×soft-time) and the user will be prompt with Yes/No dialog, as shown in Figure 11.

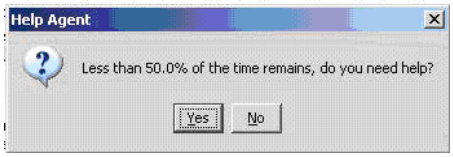

Figure 11: An example of a help dialog

The student now has the option of ignoring the agent and pressing "No" to cancel the help request, or the student can accept the extra help offered by the agent. Both responses are logged by the agent in order to adapt to the user's needs. If the student denies the help offered by the agent more than an administrator-defined number of times, the agent will become inactive and no longer interrupt the student. In this case, the student selects "Yes" and is presented with a contextual help dialog for the beginner skill level, as shown in Figure 12.

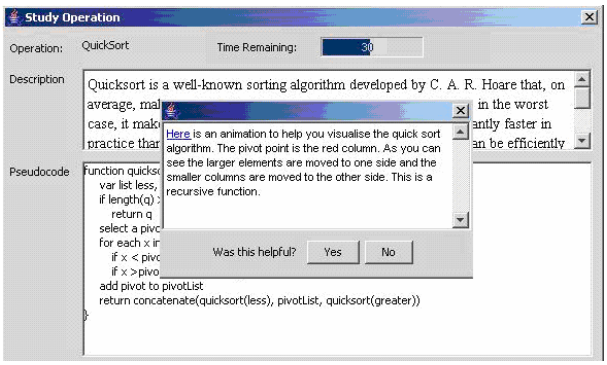

Figure 12: Beginner level help dialog

The student is now asked if this help was actually helpful in order to provide feedback for the authors to better serve students in the future. Once the time remaining falls below the soft-time for that particular lesson, the AETA warns the student that he or she may continue studying the lesson, but any extra time used will be deducted from the remaining lessons, as shown in Figure 13.

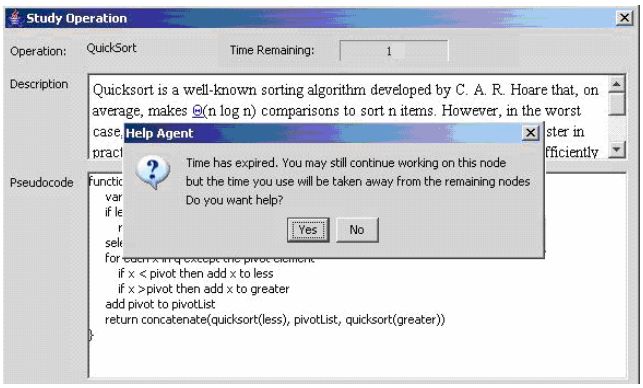

Figure 13: A dialog for time expiration

5.5 Learning

The learning component of the AETA is responsible for writing new data to the student model and the author model. The new data is gathered by the AETA as it monitors the student's progress through the lessons, algorithms, and tests. For instance, when the student completes a lesson, the time remaining for that lesson is recorded by the AETA and the student model, for this particular student, and is updated to reflect whether the student was early, late, or on time by way of the communication component. The AETA also learns the student's habits by recording the times when help is accepted and denied. If the AETA's help suggestions are denied more than some administrator-defined number of times, it will become inactive and not prompt the student in the future. The AETA can be reactivated by the student, through the SHALEX interface, which resets the denied count.

The learning of the AETA is implemented as objects in main memory which store the new information until an event caused either by the student or the timer invokes the communication component and updates the student model or the author model. The knowledge is sent from the Java application to the eXist database by XMP-RPC and is represented as an XML document. This new knowledge is integrated into the pre-existing documents which make up the two models. This approach of first storing new knowledge in main memory and then updating the knowledge base at a later time is used to cut down on the database access rate, which is a very slow process when compared with main memory access times.

6 Examples

This section describes, in detail, the possible outcomes that can result from a student using the SHALEX system and interacting with the AETA. For these examples, the SHALEX system has three student users: Student 1, Student 2 and Student 3. For our purposes, Student 1 and Student 2 have a beginner skill level, and Student 3 has an intermediate skill level**.** The algorithm explanation to be studied by these student users is the quick sort algorithm. In SHALEX, quick sort is represented by three lessons which make up the algorithm tree, as shown in Figure 14.

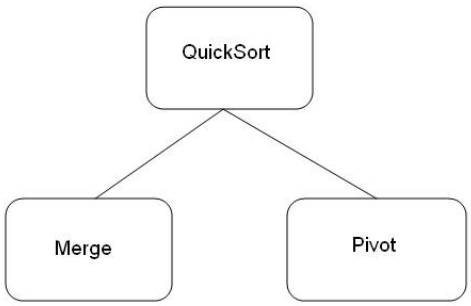

Figure 14: A tree representation of the QuickSort algorithm

The root lesson of the tree is the quick sort algorithm itself and contains a basic overview of the algorithm. The root lesson is parent to two child lessons which represent the two critical parts of the quick sort algorithm, namely: *Pivot* and *Merge*. The QuickSort, Pivot and Merge lessons have respective soft-times of 120, 60, and 60 minutes. Please note that the time unit can be changed based on the authors recommendations. In our experiments, we used minutes. The timing thresholds for the AETA interactions are scheduled for time remaining equals to 0.5 × soft-time and time remaining equals to 0.25 × soft-times.

6.1 Example 1: Student Finishes on Time

Student 1 logs into the SHALEX system and is presented with the screen shown in Figure 15.

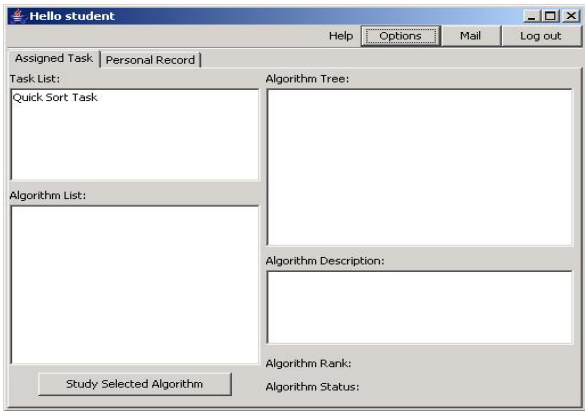

Figure 15: A dialog of available algorithms

Student 1 selects the root lesson and begins studying the “QuickSort” lesson. This lesson has a soft-time of 120 minutes. Student 1 reads through the algorithm description shown in Figure 10, clicks the hyperlinks which provide extra information about keywords, and looks over the pseudo-code for quick sort. At time remaining =

60 minutes, Student 1 has not yet completed the "QuickSort" lesson so the first warning is scheduled and sent by the AETA. Student 1 feels close to understanding the concepts presented and so denies the help suggestion by the AETA. The AETA logs this choice to deny help in Student 1's student model. Student 1 finishes studying the lesson with 20 minutes remaining. The lesson is finished before the second AETA warning and so is completed before the soft-time for the lesson expires. Note, even though there is time remaining between the time required for Student 1 to complete the lesson and the soft-time for the lesson, this time is not distributed and added to the remaining lessons (as time remaining = soft-time for these lessons) but is instead pooled for future use, if necessary. The time pool equals 20 minutes.

Student 1 now chooses to study the "Pivot" lesson which has a soft-time of 60 minutes. Student 1 studies the lesson but has trouble understanding from the description and pseudo-code alone. When the time remaining reaches 30 minutes, the warning is scheduled and sent by the AETA, as shown in Figure 16.

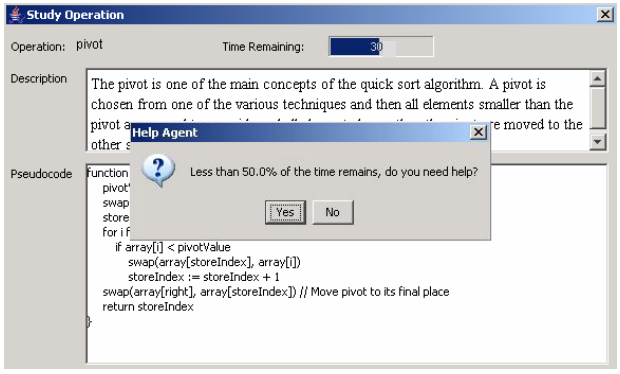

Figure 16: The first warning sent by AETA for the Pivot lesson

This time, Student 1 accepts the offer for extra help and is presented with an extra help dialog. The time remaining continues to decrease as Student 1 views the extra help dialog. Student 1 studies the extra help and clicks the hyperlink which opens Student 1's default web browser and displays a helpful image shown in Figure 17. This image is from (http://www.cosc.canterbury.ac.nz/research/RG/alg/quicksort.gif).

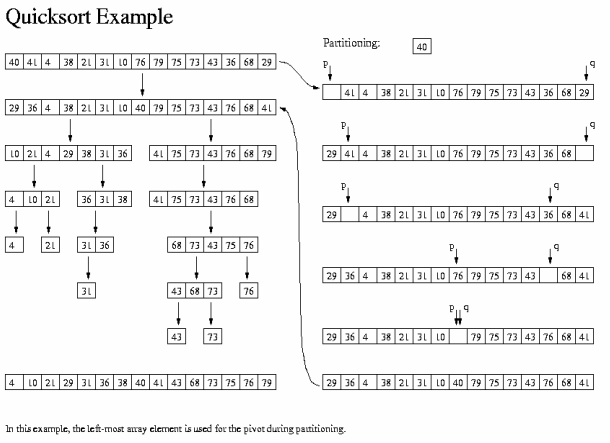

Figure 17: A Quicksort image accessed through a web browser

As Student 1 is looking at this image and relating it to the written help, the time remaining reaches 15 minutes and the minute warning is scheduled, but is not displayed since the help window is already open. Student 1 shuts the help dialog and proceeds to reread the algorithm description and apply the new knowledge from the help dialog. The time remaining reaches zero minutes and the final warning is scheduled and sent, which instructs Student 1 that time has expired and any more time spent studying the lesson will be deducted from the remaining lessons (in this case only the "Merge" lesson). Help is once again offered to Student 1. Again, Student 1 accepts the help and this time fully understands the "Pivot" lesson. Even though Student 1 finishes the "Pivot" lesson 10 minutes late, the extra time used is first deducted from the pooled time (of 20 minutes) and therefore the time remaining for the "Merge" lesson remains equal to its soft-time of 60 minutes. Student 1 chooses the final lesson called "Merge." When the time remaining reaches 30 minutes, the AETA schedules and sends the first warning. Student 1 decides to deny the help suggestion as he or she is not completely finished reading all of the information. At time remaining = 15 minutes, the second warning is scheduled and sent, as shown in Figure 18.

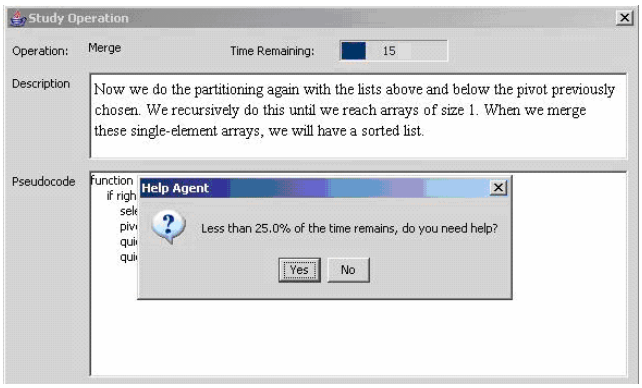

Figure 18: AETA displays the second warning for Merge lesson

Student 1 accepts this help suggestion, and the extra help dialog is displayed. Using this extra help, Student 1 understands the "Merge" lesson and completes the lesson. Now, all three lessons are complete and, consequently, the algorithm is complete. Even though the "Pivot" lesson was completed late, the overall algorithm was completed on time. Figure 19 shows the AETA informing the student of her completion. This is recorded and stored in the student model for future use.

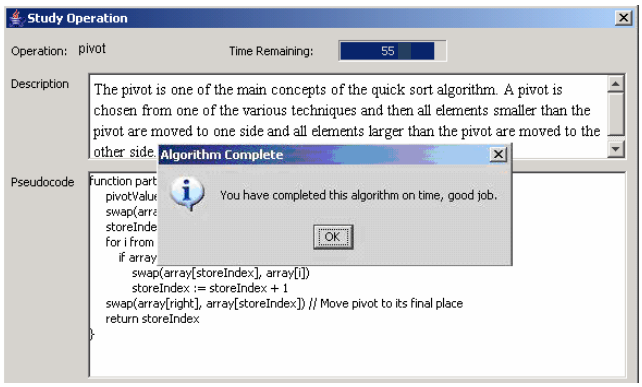

Figure 19: AETA informs the student of the completed algorithm

6.2 Example 2: Student Finishes Late

Student 2 logs into the SHALEX system and feels he learns better with a bottom up approach rather than a top down approach so he selects one of the children lessons to study first. Student 2 selects the "Merge" lesson and begins learning by viewing the description and the pseudo-code. Since the "Merge" lesson is a child of "Pivot" lesson, the author intended the "Pivot" lesson to be studied before the "Merge" lesson. Student 2 has problems understanding the "Merge" lesson and accepts help on both warnings issued by the AETA at time remaining = 30 minutes and time remaining = 15 minutes, shown in Figure 20**.**

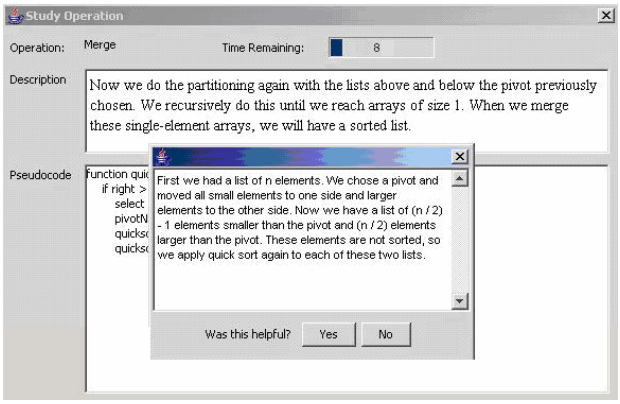

Figure 20: The help dialog for the Merge lesson

Time finally expires and the AETA schedules and sends the final warning message similar to the message shown in Figure 13. Student 2 views the help one last time and finishes the lesson 10 minutes late. Since this was the first lesson attempted by Student 2, the extra time pool is at zero minutes, therefore the 10 minutes of extra time needed by Student 2 is removed from the remaining two lessons. Student 2 continues with the bottom-up philosophy and selects the "Pivot" lesson next. The "Pivot" lesson begins with 5 minutes already removed from the time remaining because the 10-minute penalty was split between the "Pivot" and "QuickSort" lessons, as shown in Figure 21.

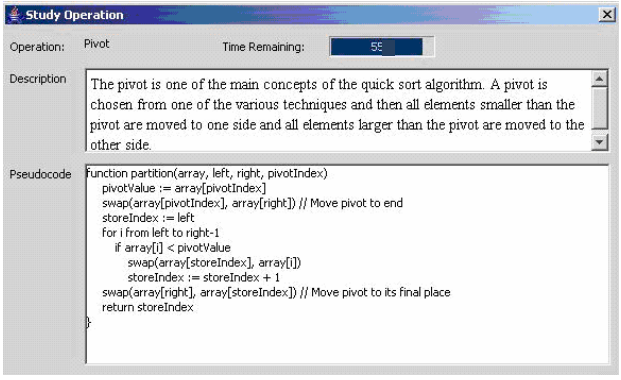

Figure 21: Five minutes deducted from Pivot lesson's soft-time

As was the case with the "Merge" lesson, Student 2 is warned twice by the AETA and finally the time expires. Student 2 accepts the offer for help and finishes the "Pivot" lesson 15 minutes after the allowable soft-time. Student 2 selects the "QuickSort" lesson as the only remaining choice. The lesson begins with 20 minutes removed from

the time remaining (5 minutes from the "Merge" lesson time violation + 15 minutes from the "Pivot" lesson time violation), as can be seen in Figure 22.

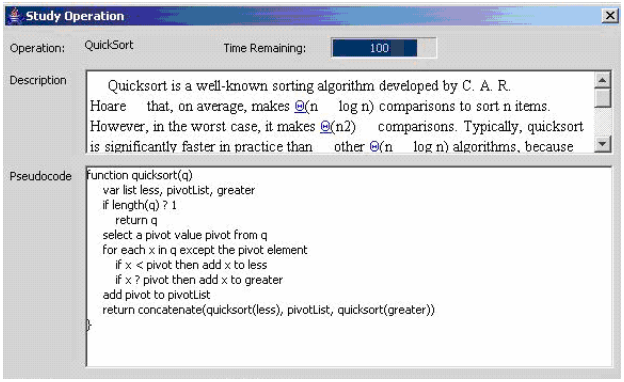

Figure 22: Twenty minutes deducted from the QuickSort lesson

As the time remaining reaches 60 minutes, the first warning from the AETA is scheduled and sent. Student 2 chooses to deny the help at this time. When the time remaining reaches 30 minutes, the second warning from the AETA is scheduled and sent, and this time Student 2 chooses to view the extra help dialog. The lesson time expires before Student 2 finishes studying and therefore he is deemed late, as shown in Figure 23. This is recorded in Student 2's student model for future use.

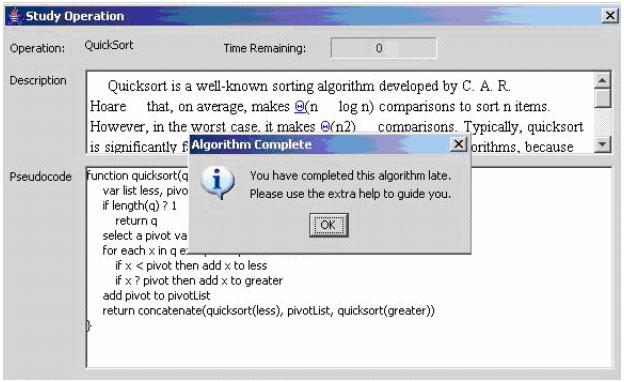

Figure 23: The AETA informs the student that he is late

6.3 Example 3: Student Finishes Early

Student 3 logs into the SHALEX system and begins by choosing the "Quick Sort Task" from the task list. This displays the "QuickSort" algorithm in the Algorithm List. Student 3 highlights this choice and proceeds to click the "Study Selected Algorithm" button. Choosing the top-down approach, Student 3 begins by studying the "QuickSort" lesson. After 30 minutes has expired, the AETA prompts Student 3

and asks him if he requires help. Student 3 accepts and is presented with the help dialog in Figure 24. This help is different from the help received by Student 1 and Student 2, because Student 3 is of a higher skill level.

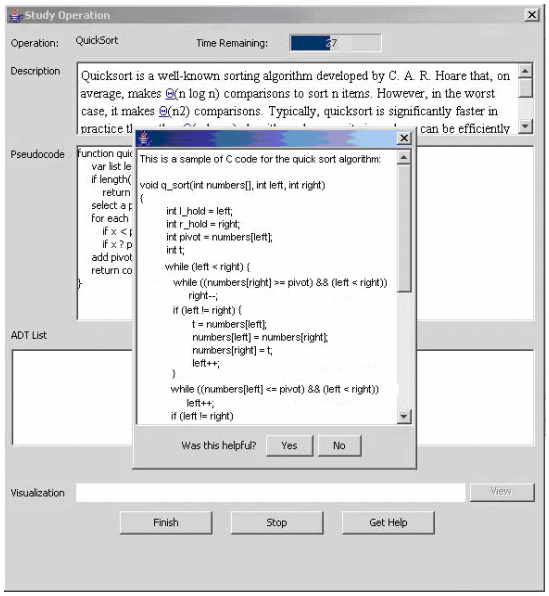

Figure 24: A help dialog for the intermediate skill level

After reviewing this help, Student 3 feels confident in his understanding of the "QuickSort" lesson and clicks the "Finish" button. Student 3 finishes the lesson with 27 minutes remaining so this time is banked in the time pool. Student 3 proceeds to examine the "Merge" and "Pivot" lessons, respectively. Since Student 3 is of a higher skill level, he finishes all three lessons before the time remaining expires. The time pool was not needed by Student 3, in this case, so it is discarded as it offers no benefit to carry extra time from one algorithm to the next. Student 3 is presented with the message in Figure 19 and his successful early completion is recorded in the student model.

6.4 Example 4: Student Taking a Test

Upon the completion of the "QuickSort" algorithm, Student 3 is required to finish the test associated with "QuickSort." After closing the dialog explaining his early completion of the algorithm, Student 3 is presented with the screen shown in Figure 25.

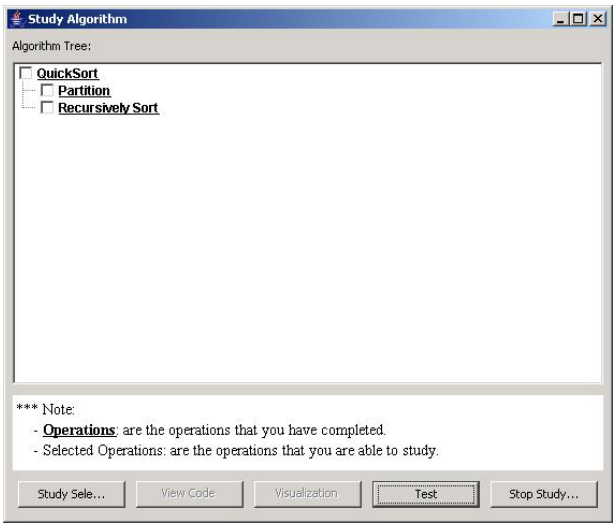

Figure 25: A dialog showing the completed algorithm

The unchecked boxes in Figure 25 represent completed lessons. Since there are no checked boxes, Student 3 has completed the algorithm and is free to take the test. Student 3 clicks on the "Test" button and the screen pictured in Figure 26 is displayed.

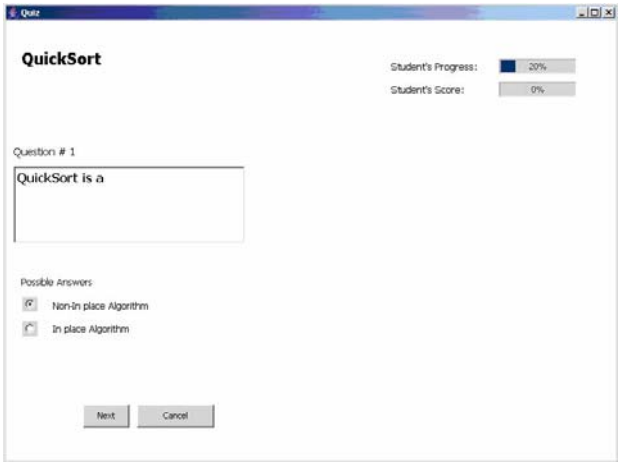

Figure 26: The initial test screen

The test dialog has two status bars. The top bar depicts the student's progress through the test. Here, the student is on question 1 of 5, so is 20 percent finished. The second bar shows the student's score. This is updated whenever the student answers a question correctly. The "Next" button is used to submit the currently selected choice

and bring up the next question. The "Cancel" button is used to stop taking the test with the expectation the student will resume at a later time. To avoid cancelling a test, reading up on the question and resuming the test later, the currently selected choice is taken to be a final answer for that question. In this case, the student is warned before the answer is submitted.

Student 3 knows the first answer is correct and clicks the "Next" button. Figure 27 shows that the student's score and progress have both been updated and the second question is then presented.

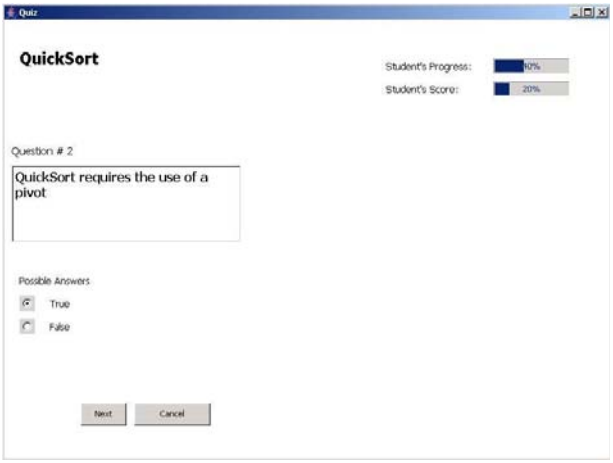

Figure 27: An updated test display showing the second question

Student 3 knows the answer to this next question is "True" so he clicks "Next" and is presented with question 3, shown in Figure 28.

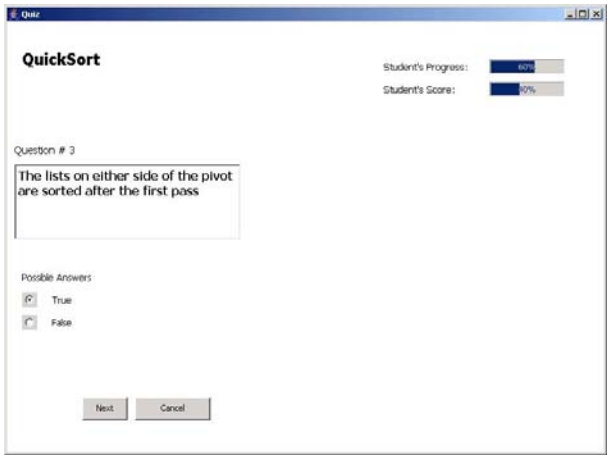

Figure 28: A test dialog showing the third question

So far, Student 3 has answered both questions correctly, which is reflected in the student's score. This time, Student 3 is not sure of the answer and he again chooses "True." This is not the correct answer and Figure 29 shows the score has remained the same while the test question and overall progress has been updated.

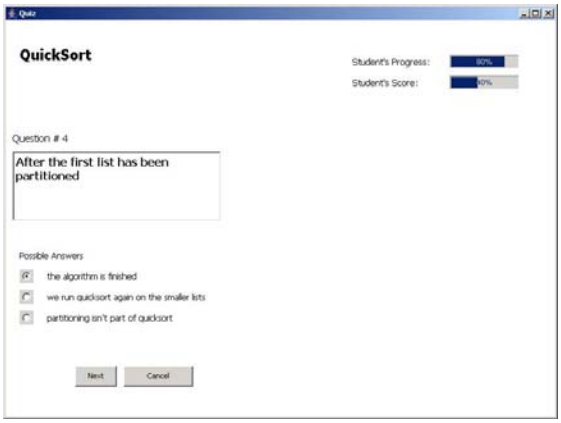

Figure 29: A test dialog showing the fourth question

Student 3 again makes a wrong choice on question 4 and is shown the final question. Student 3 makes a final incorrect choice and the test is finished. Figure 30 shows the final dialog presented by the test which tells Student 3 of their final mark. This final score is stored in Student 3's model and is used to re-evaluate the user's skill level in the future.

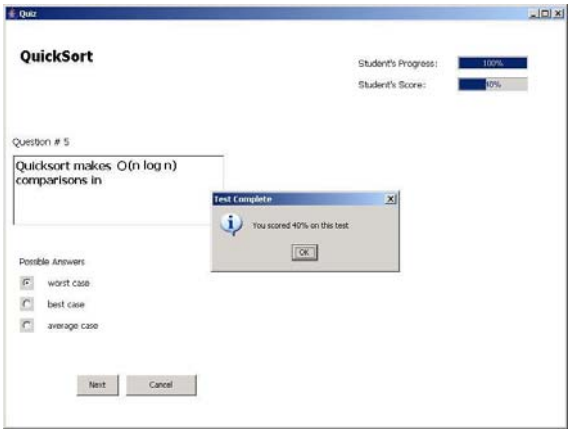

Figure 30: A dialog showing the final test score

7 Conclusions and Future Work

This paper has presented an intelligent interface agent which helps guide students to an understanding of algorithms within the domain of SHALEX. The AETA uses the

knowledge of an ideal learning model in order to schedule a proper sequence of actions and provide assistance, where necessary, in order to help students.

Each student is assigned his or her own agent, but the information gathered by each of these agents and stored in the student's model is never shared. Even though multiple agents exist on the system, there is no collaboration among agents so this is a single-agent architecture. In future work, it would be beneficial to implement a multi-agent architecture so the agents can collaborate and share knowledge of the users.

Acknowledgements

The work is supported by a grant from the Natural Sciences and Engineering Research Council of Canada (NSERC) and internal grants from Acadia University. The authors would like to sincerely thank Dr. Müldner for his help and support during all stages of this project. The authors would also like to thank two alumni from Jodrey School of Computer Science for their contributions to this project; Joe Merrill for his implementations in Macromedia Flash MX during the first stages of the project, and Brad Haughn for his implementations of some parts of the proposed system.

References

[Baker, 04] Baker, R. S., Corbett, A. T., Koedinger, K. R., Wagner, A. Z.: "Off-task Behavior in the Cognitive Tutor Classroom: When Students "Game the System" "; ACM Proceedings of CHI, Vienna, Austria (2004), 383-390.

[Braune, 00] Braune, B., Wilhelm, R.: "Focusing in Algorithm Explanation"; IEEE Transactions on Visualization and Computer Graphics (2000), 6, 1, 1-7.

[CIMEL, 06] CIMEL: Constructive, collaborative, Inquiry-based Multimedia E-learning, http://www.cse.lehigh.edu /~cimel/, Computer Science and Engineering Departments, Lehigh University, Bethlehem, PA, USA, 2006.

[Curcio, 98] Curcio, F. and Schwartz, S.: "There are no Algorithms for Teaching Algorithms"; Teaching Children Mathematics, Reston, 5, 1(1998), 26-30.

[Demmings, 07] Demmings, B., Shakshuki, E., Müldner, T.: "AETA: Algorithm Explanation Teaching Agent"; Proc. International Conference on Information Technology: New Generations (ITNG), IEEE Computer Society, April 2-4, Las Vegas, USA (2007), 616-621.

[Dijkstra, 89] Dijkstra, E.W.: "On the Cruelty of Really Teaching Computing Science"; SIGCSE Award Lecture, Communications of the ACM, 31, 12(1989), 1398-1404.

[Dudek, 08] Dudek, D.: "The APS Framework for Incremental Learning of Software Agents"; Journal of Universal Computer Science, 14, 14(2008), 2263-2287.

[eXist, 08] eXist. Open Source Native XML Database, 2008, http://exist.sourceforge.net/

[Fok, 06] Fok, A., Ip, H.: "An Agent Based Framework for Personalized Learning in Continuing Professional Development"; International Journal of Distance Education Technologies, 4, 3(2006), 48-61.

[Flahive, 05] Flahive, A., Taniar, D., Rahayu, W.: "Database Design for Web-Based Product Catalog"; International Journal for Computers and Their Applications, International Society for Computers and Their Applications, USA, ISBN/ISSN: 1076-5204, (2005), 78-91.

[Grissom, 03] Grissom, S., McNally, M. & Naps, T.: "Algorithm Visualization in CS Education: Comparing Levels of Student Engagement"; ACM Symposium on Software Visualization, New York: ACM (2003), 87-94.

[Hübscher-Y., 03] Hübscher-Younger, T., Narayanan N. N.: "Dancing Hamsters and Marble Statues: Characterizing Student Visualizations of Algorithms", ACM Symposium on Software Visualization, San Diego, California, USA (2003), 95-104.

[Kutay, 05] Kutay, C., Ho, P.: "Designing Agents for Feedback Using the Documents", International Journal on E-Learning, Norfolk, VA: AACE, 4, 1(2005), 99-109.

[Moritz, 05] Moritz, S., Wei, F., Parvez, S., Blank, G.: "From Objects-First to Design-First with Multimedia and Intelligent Tutoring", ACM Proceedings of ITiCSE, June 27-29, Monte da Caparica, Portugal (2005), 99-103.

[Müldner, 06] Müldner, T. and Shakshuki, E.: "Explaining Algorithms: A New Perspective"; International Journal of Distance Education Technologies. Idea Group Inc., 4, 3(2006), 6-23.

[Müldner, 08] Müldner, T., Shakshuki, E., Kerren, A.: "Algorithm Education Using Structured Hypermedia"; Strategic Applications of Distance Learning Technologies, Idea Group Inc., Chapter V (2008), 58-84.

[Naps, 03] Naps, T. L., Rolling, G., Almstrum, V., Dann, W., Fleischer, R., Hundhausen, C., Korhonen, A., Malmi, L., McNally, M., Rodger, S., and Velazquez-Iturbide, J. A.: "Exploring the Role of Visualization and Engagement in Computer Science Education"; SIGCSE Bulletin 35(2003), 131-152.

[Shakshuki, 04] Shakshuki, E., Müldner, T., Haughn, B.: "ATIA: Algorithm Teaching Interface Agent"; Proceedings of the 5th International Conference on Information Technology Based Higher Education and Training (ITHET'04), IEEE Computer Society, May 31st-June 2nd, Istanbul, Turkey (2004), 138-143.

[Shakshuki, 07] Shakshuki, E., Kerren, A., Müldner, T.: "Web-based Structured Hypermedia Algorithm Explanation System"; International Journal of Web Information Systems, Emerald Publisher, 3, 3(2007), 179-197.

[Shakshuki, 08] Shakshuki, E., Halliday, R.: "An Algorithm Explanation Agent for the SHALEX System"; the 10th International Conference on Information Integration and Web-based Applications & Services (iiWAS), ACM, November 24-26, Linz, Austria (2008), 292-298.

[Shi, 00] Shi, H., Shang Y., Chen, Su-Shing.: "A Multi-Agent System for Computer Science Education"; Proceedings of the 5th annual SIGCSE/SIGCUE conference on Innovation and Technology in Computer Science Education (ITiCSE), Helsinki, Finland (2000), 1-4.

[Vassileva, 03] Vassileva, J., McCalla, G., Greer, J.: "Multi-Agent Multi-User Modeling in I-Help"; User Modeling and User-Adapted Interaction, Springer, 13, 1-2(2003), 179-210.

[Wei, 05] Wei, F., Moritz, S., Parvez, S., Blank G.: "A Student Model for Object-Oriented Design and Programming"; Journal of Computing Sciences in Colleges, 20, 5(2005), 260-273.

[Zaupa, 08] Zaupa, F., Gimenes, I., Cowan, D., Alencar, P., Lucena, C.: "A Service-oriented Process to Develop Web Applications"; Journal of Universal Computer Science, 14, 8(2008), 1368-1387.