Web-based Learning through Mixed-Initiative Interactions: Design and Implementation

Nantha Kumar Subramaniam

Open University Malaysia, Jalan Tun Ismail, 50480 Kuala Lumpur, Malaysia

nanthakumar@oum.edu.my

Tel: 6-03-27732114

Accepted subtheme: **3**

Abstract*: Mixed-initiative interaction is a naturally-occurring feature of human-human interactions. It characterize by turn-taking, frequent change of focus, agenda and control among the "speakers". This human-based mixed-initiative interaction can be implemented through a mixed-initiative systems which are a popular approach to building intelligent systems that can collaborate naturally and effectively with people. Mixed-initiative systems exhibit various degrees of involvement in regards to the initiatives taken by the user or the system. In any discourse, the initiative may be shared between either, a learner and a system agent, or between two independent system agents. Both the parties in question establish and maintain a common goal and context, and proceed with an interaction mechanism involving initiative taking that optimizes their progress towards the goal. However, the application of mixed-initiative interaction in web-based learning is very much limited. In this paper, we discuss the design and implementation of a web-based learning system through mixed-initiative system known as JavaLearn. JavaLearn allows the interaction between the system (in the form of a software agent) and the individual learner. Here, the system supports the learning through a problem solving activity by demanding active learning behaviour from the learner with minimal natural language understanding by the agent and embodies the application-dependent aspects of the discourse. It guides the learner to solve the problem by giving adaptive advice, hints and engage the learner in the real time interaction in the form of "conversation". The principal features of this system are: It is adaptive and are based on reflection, observation and relation. The system acquires its intelligence through the finite state machine and rule-based agents.*

Introduction

Four mode of interactions that can be found in a web-based learning environment are **student-student**, **student-instructor**, **student-content** and **student-interface** (Thurmond & Wambach 2004). The **student-interface** interaction is a new form of interaction, thanks to the increased processing power of computers and the advancement made in the field of artificial intelligence (Thurmond & Wambach 2004). The student-interface interaction is defined as the interaction between the learner and the tools needed to perform the required learning task. In most cases, student-interface requires active participation from the user. This eventually demands active learning behaviour among the learners. Furthermore, active learning is an important teaching and learning technique especially for the adult learners (Huang 2002).There are various ways that can be adopted to realize the student-interface interaction. One way to do it is through mixed-initiative interaction. Mixed-initiative interaction is a naturally-occurring feature of human-human interactions (Menon et al. 2005).

It is characterized by turn-taking, frequent change of focus and control among the “speakers”. This human-based mixed-initiative interaction can be implemented through a mixed-initiative systems which are a popular approach to building intelligent systems that can collaborate naturally and effectively with people. Mixed-initiative systems exhibit various degrees of involvement in regards to the initiatives taken by the user or the system. In any discourse, the initiative may be shared between either, a learner and a system agent, or between two independent system agents. Both the parties in question establish and maintain a common goal and context, and proceed with an interaction mechanism involving initiative taking that optimizes their progress towards the goal. One of the key elements for successful mixed-initiation is the ability of the system to recognize opportunities for mixed-initiative interactions.

Problem Statement

The use of mixed-initiative interaction in the form of mixed-initiative systems for web-based learning is considered a new phenomenon as most of the mixed-initiative systems have been developed for non-education sector (Rich & Sidner, 1998). Limited researches of using mixed-initiative systems for education purposes have been carried out by Hanson, Judd and Rich (2009), Shakya et al. (2005) and Rao et al. (2006) especially in the programming courses. Hanson, Judd and Rich (2009) had designed a game environment to teach students about basic programming and object-oriented concepts using text-based mixed-initiative interaction. Shakya (2009) used Self Regulated Learning to determine strategies and tactics that learners used in their mixed-initiative interactions. The system was modeled based on the concept of pair-programming. On the other hand, Rao et al. (2006) had developed a real-time architecture called MICE (Mixed-Initiative Coding Environment). It uses ontologies to model-trace programming styles, employs rules to assist programmers to regulate their programming styles, and engages mixed-initiative scaffolding tactics and strategies to provide feedback. In open and distance education, web-based learning is normally conducted using Learning Management System such as WebCT or Moodle and these platforms lack the mixed-initiative interaction component.

Objective

The objective of the paper is to discuss the design and implementation of web-based learning system based on mixed-initiative system for learning of Java programming. Java is chosen as it is a programming subject that requires active learning approach in order to understand the subject matter. The system/prototype known as *JavaLearn* is able to engage the learners in a problem solving activity and at the same time allows the students to “interact” with the system as part of the learning process activity in solving the problem. Here, the system supports the learning through a problem solving activity by demanding active learning behaviour from the learner with minimal natural language understanding by the agent and embodies the application-dependent aspects of the discourse. It guides the learner to solve the problem by giving adaptive advice, hints and engage the learner in the real time interaction in the form of “conversation”. The principal features of this system are: it is adaptive and are based on reflection, observation and relation.

Prototype Design

Our version of the software agent paradigm which we term *JavaLearn* is illustrated in Figure 1. This paradigm mimics the relationships that hold when two humans collaborate on a task involving a shared artifact, such as two mechanics working on a car engine together or two computer users working on a spreadsheet together. Notice that the software agent is able to both communicate with and observe the actions of the user and vice versa. A crucial part of successful collaboration is knowing when a particular action has been performed. In our proposed *JavaLearn*, this can occur one way: by clicking the action button. Typically, the agent queries the application state using the finite state machine. The tasks that can be done by the user in *JavaLearn* are:

i. Solve the problem by doing the action in an orderly manner represented as "cycles" (described later in other section)

ii. While solving the problem, chat with the agent using the "predefined messages" provided by the agent. Although, in the long run, communication between users and interface agents will very likely be in spoken natural language, we have decided to include limited natural language understanding in *JavaLearn*. As a practical matter, natural language understanding, even in this limited setting, is a very difficult problem in its own right, which we would like to sidestep for the moment.

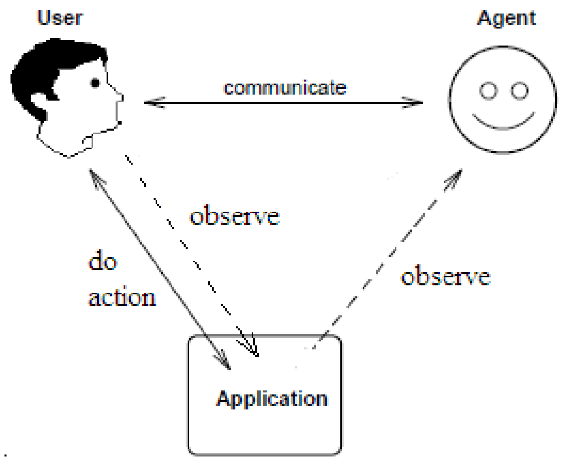


Figure 1 Collaborative interface agent for *JavaLearn*

This discussion between the agent and student in *JavaLearn* is conducted on the premise of that the learners' learning is not so much a matter of building up correct responses or eliminating incorrect responses. The most important thing is for students to have the opportunity to test the adequacy of their ideas. It is the process of how the learners "*persist*" in the problem solving activity rather than on actually being able to solve the problem successfully. The general overview of the system (*JavaLearn*) is shown below.

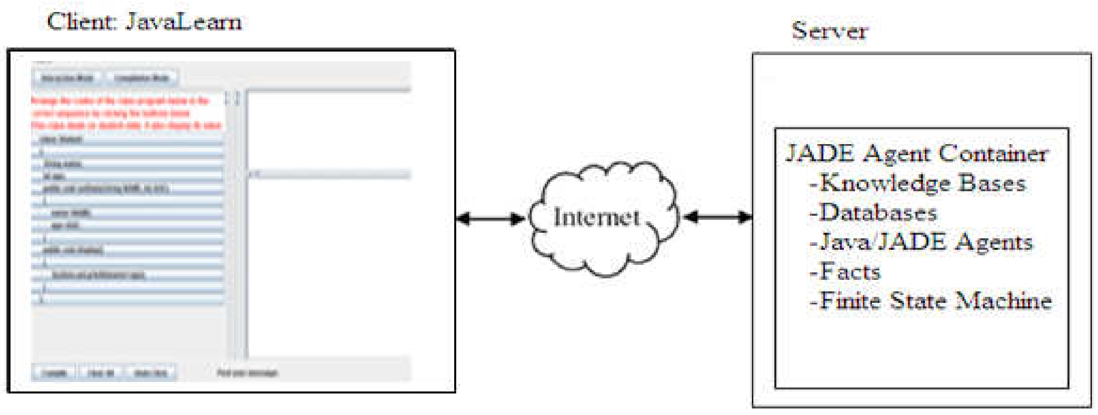


Figure 2 General Overview of the *JavaLearn*

The proposed architecture of the system will use rule-based multi-agent approach. Agent approach is adopted as it is goal oriented, take action when necessary to fulfil the goal, capable to perform tasks given by the user autonomously, monitor the environment and adjust an event without direct intervention from the user. Figure 2 shows the components that make up the proposed system. *JavaLearn* has FOUR agents, namely **chat**, **helper**, **advisor** and **pedagogy** agents performing different tasks. The facts and rules for the agents will be stored in the respective knowledge bases. In *JavaLearn*, the students are given a task or problem to be solved through collaborative discussion with the **chat** agent. In order to engage in the discussion, the students will post their messages using the “predefined messages”. These “predefined messages” are determined by the system during the runtime based on the user’s state in the finite machine. Only one message can be selected per posting by the learner to engage in the discourse. These “predefined messages” for the learners to choose are in the form of “questions” as listed below (Rich et al. 2001):

What ...
Where ....
When ....
Why ....
How...
Can ....

In addition, the following “predefined messages” are also included in *JavaLearn* so that the learners can take a lead in the discourse with the agent irrespective whether the learner is on the right track in solving the problem. This is in line with the mixed-initiative philosophy that mimics human-human interaction that has frequent change of focus:

No..I do not think so
Why not ….
I think ….
We should ….

We are motivated to use the “predefined” messages based on the work done on using sentence opener (Baker & Lund (1996). In the sentence opener approach, the opening words is given and student need to complete the sentence using their own words. However, in

JavaLearn, the complete sentences are given for the students to choose and they are not required to type any extra words. This will reduce the mental load of the student in solving the problem given to them. In this study, the "predefined messages" are formulated based on the Collaborative Skills Network (CSN) proposed by Israel (2003). The state of the finite machine in *JavaLearn* will determine which messages that will be pushed for the students view for that particular problem solving cycle. Figure 3 shows the *JavaLearn* architecture.

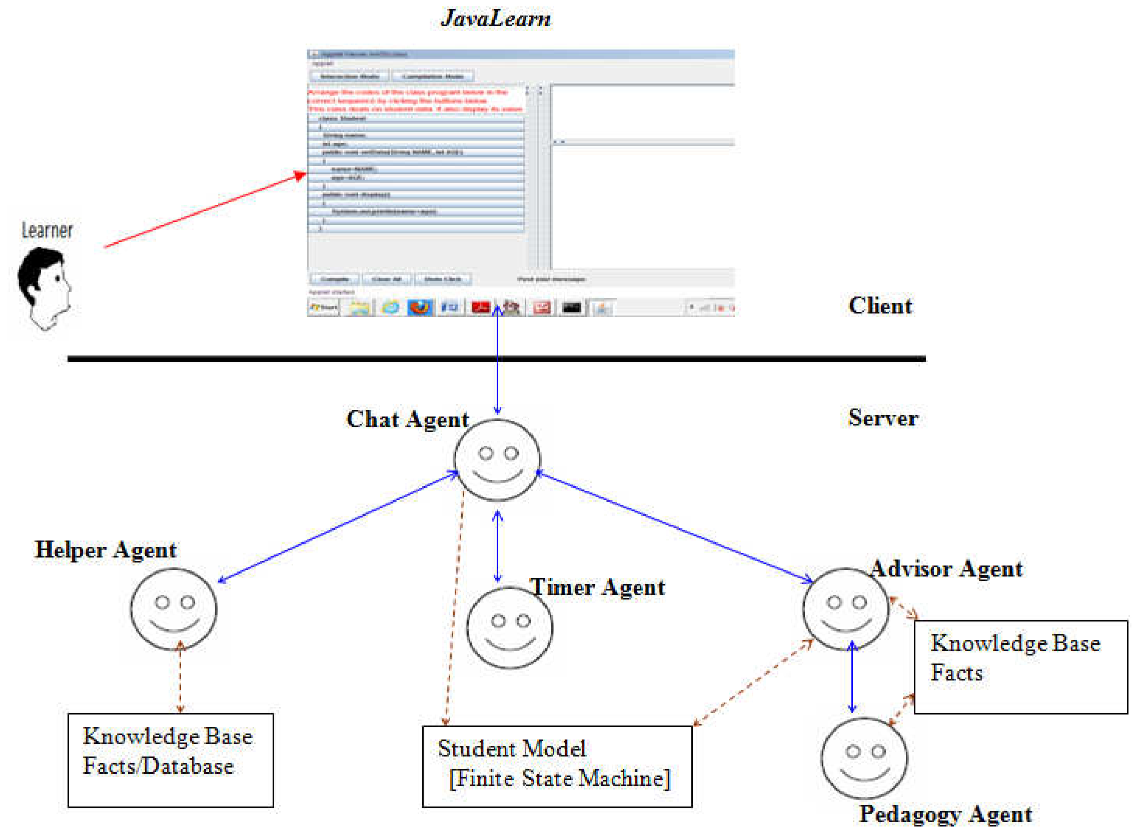


Figure 3 *JavaLearn* architecture

In our proposed system (*JavaLearn*), each action selected by the learner will be first parsed by the ***chat*** agent that will do the following tasks as describe in Table 1.

Table 1 Assistant agents in *JavaLearn*

i.	Identify whether the student has posted the message using the pre-defined message provided the system or has used his/her own words. If latter is the case, the **chat** agent will invoke the ***helper*** agent;
ii.	Identify if the student is yet to post any new message after certain time interval and call the ***timer*** agent if it is so; and
iii.	If the student has posted a message from the predefined messages prepared by the system within the time frame, it will call the ***advisor*** Agent.

In the case of (i), **helper** agent will analyse the message posted by the student and select the appropriate response(s) from the knowledge base. This response(s) will be sent to the **chat** agent so that it can be posted in the chat interface for the student view. If the **helper** agent fail to understand the message posted by the user, it will advise the learner (via **chat** agent) to select the predefine message already provided by the system. This agent uses *Knuth-Morris* pattern-matching algorithm in analyzing the messages posted by the learner.

In the case of (ii), the **timer** agent will send an alert message via **chat** agent reminding the students that he/she spending too much of the time in selecting the “action”.

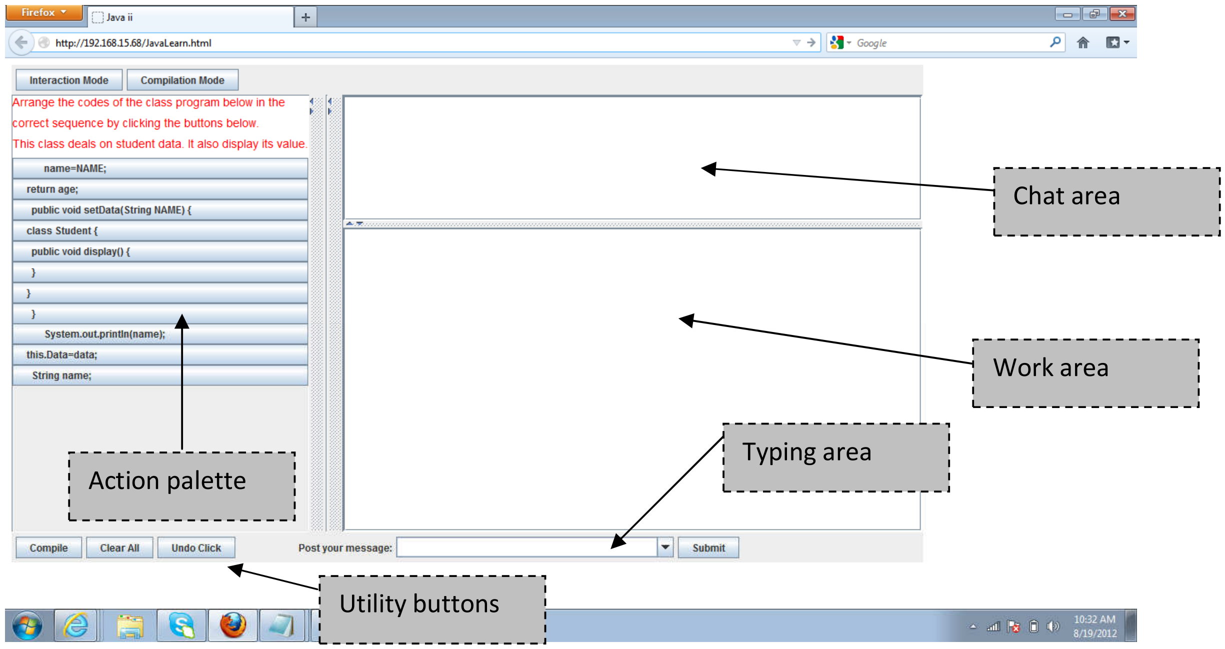
In the case if (iii), **advisor** agent will determine the suitable reply that need to be sent to the **chat** agent by querying the current state of the finite machine. the At the same time, if the learner has selected the wrong action and struggling to get the correct “action” in a particular cycle, the agent will call **pedagogy** agent in order to give an appropriate advice to the student. This is done by utilizing the knowledge base and finite state machine. When the learner is in the midst of solving the problem by selecting the correct “action”, the **chat** agent will also update the finite state machine to reflect the current student model. In all these cases, **chat** agent merely acts as an interface between the system and the user. It conveys the message posted by the user to other agent and at the same time, update the finite machine. **Chat** agent uses the identity “DrJava” when posting the messages. At one time, the **chat** agent can only call one agent. Since there are multi-agents in *JavaLearn*, **chat** agent will use the following priority level (from Table 1) to determine which agent that need to be called:

Priority level for the **chat** agent: `ii>i>iii`

Finite state machine is used to (a) keep track the sequence of action selected by the user; and (b) control the flow of conversation. Users must select the message from the list each time they add to the discussion. The list is determined by the state in the finite state machine which provides a mechanism to structure, and rather than to understand, the conversation. Finite state machine is elaborated in detail in the next section.

Prototype Implementation

The agents in the *JavaLearn* have been built using JADE (Java Agent Development Environment) while the interface is a Java Applet. These agents will involve in back-end processing running in a LINUX server. It can be called from the web browser by typing its URL. Figure 4 shows the interface of *JavaLearn*.

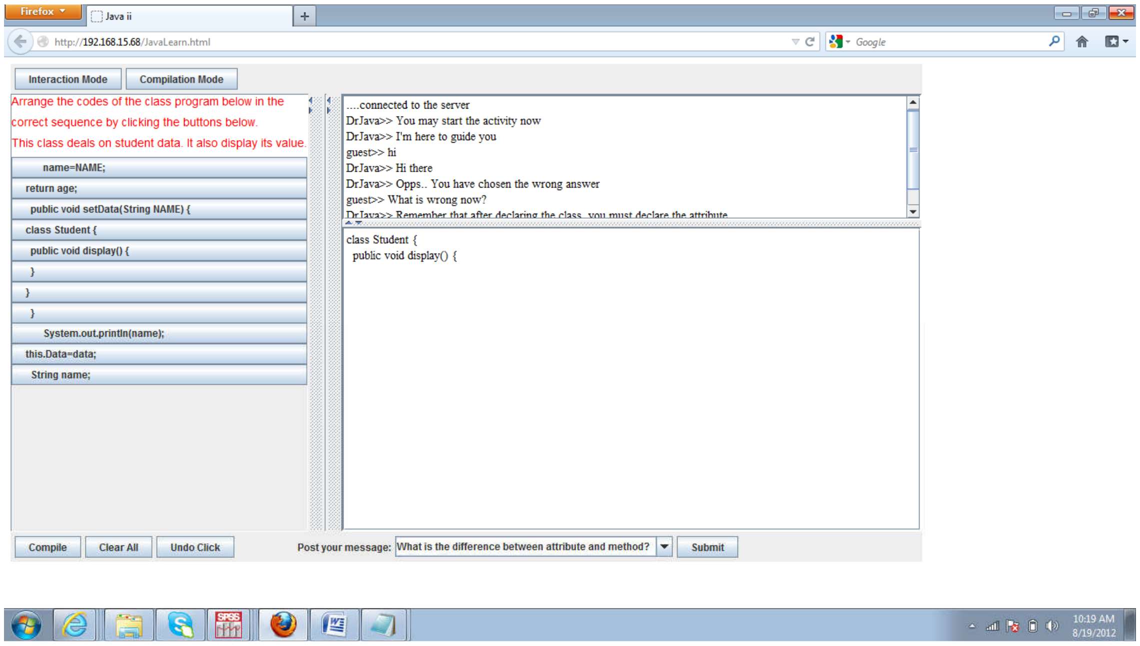


Arrange the codes of the class program below in the correct sequence by clicking the buttons below.
This class deals on student data. It also display its value.

```
      name=NAME;
  return age;
   public void setData(String NAME) {
  class Student {
   public void display() {
   }
  }
   }
      System.out.println(name);
  this.Data=data;
   String name;
```

Figure 4 *JavaLearn* interface

Chat area is the place where the learner will interact with the agent. **Work area** is the place where the student will construct the class program by choosing the correct "action buttons" from the action palette. Here, the buttons represents "actions" that can be chosen by the student. Each action has its own identifier. **Typing area** is the editable combo box where the student will type their queries or select predefined messages provided by the system. **Action Palette** contains the disorganised program codes. The learner needs to arrange the codes in the correct sequence in the work area so that it forms a complete class program (Note: Developing class program is the most important concept in Java). This is done by clicking the button (which represent the "action") and it will get displayed in the work area of *JavaLearn*. When the learner is in the midst of solving the problem by selecting the correct "action" from the action palette, it will be updated in the finite state machine to reflect the current student model. Figure below shows a segment of an example of real usage of *JavaLearn*:



Arrange the codes of the class program below in the correct sequence by clicking the buttons below.
This class deals on student data. It also display its value.

```
      name=NAME;
return age;
  public void setData(String NAME) {
class Student {
  public void display() {
  }
}
  }
      System.out.println(name);
this.Data=data;
  String name;
```

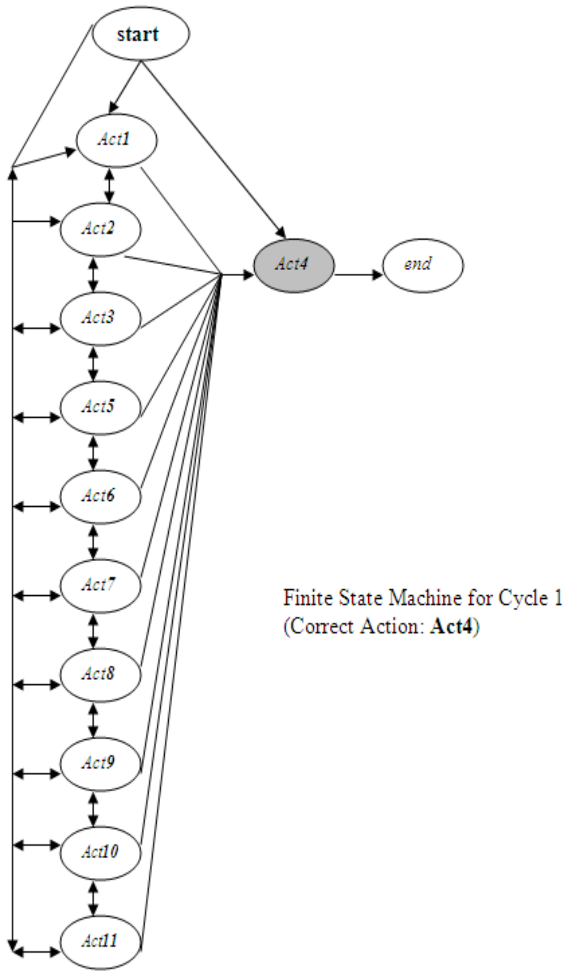
```
....connected to the server
DrJava>> You may start the activity now
DrJava>> I'm here to guide you
guest>> hi
DrJava>> Hi there
DrJava>> Opps.. You have chosen the wrong answer
guest>> What is wrong now?
DrJava>> Remember that after declaring the class, you must declare the attribute
```

```
class Student {
  public void display() {
```

Post your message: What is the difference between attribute and method?

Figure 5 A segment of interactions in *JavaLearn* between the student (*guest*) and the agent (*DrJava*)

In the case of wrong action selected by the learner, the system is capable to revert to its earlier state so that the student can resume working to select the correct action. There are 11 possible actions that can be chosen by the learners for the one problem given in *JavaLearn* (Figure 5). Among these 11 actions, 9 is the correct actions and 2 actions are the distracters. Thus, there will be 9 problem solving cycles for this problem. In each cycle, the learners may select the correct answer in the first attempt or obtain the correct answer after few attempts. In either cases, the **chat** agent will guide the learners until he/she chooses the correct answer so that they can progress to the next cycle. Figure 6 shows the possible paths in the finite state machine (FSM) for cycle 1. Figure 7 shows the possible paths in FSM for last cycle.



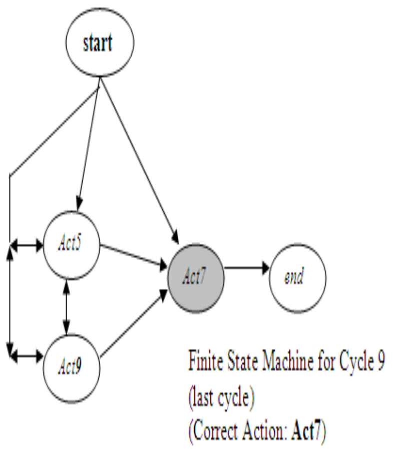


Figure 7 FSM for the last cycle

Figure 6 FSM for 1st cycle

Note: Act*n* in Figure 6 and Figure 7 refers to the "action button" identifier in the action palette

The depth/density of FSM will be reduced as a student progressing through these cycles. This is because the available actions that can be selected by the learners will be reduced as they progressing over the problem solving cycles. The agent will "fire" the appropriate feedbacks/messages/hints based on the paths taken by the student in the finite state machine. In each of these cycles, the control of the discussion may shift alternately between the learner and the agent. This is in line with philosophy of the mixed initiative interactions which mimics human-human interaction.

Learners' Evaluation

The beta-testing has been conducted for *JavaLearn* by 5 learners who took this course. At the end of using the system, a questionnaire has been distributed to them. The questionnaire has 8 items and are measured in the Likert scale of 1 (very weak) to 5 (very good). The mean scores for all the items are shown below.

Table 2 Mean score of the items

	Item	**Mean Square**
1	How would you rate the accuracy of *JavaLearn*? (accuracy refers to the correctness of the responses displayed by *JavaLearn*)	**3.33**
2	How would you rate the usefulness of the "predefined messages"?	**3.50**
3	How would you rate the quality of the responses made by *JavaLearn*? Quality refers to clarity and appropriates of messages responded by *JavaLearn.*	**3.75**
4	How would you rate the usefulness of *JavaLearn* for accomplishing your individual work?	**3.63**
5	How would you rate the usefulness of *JavaLearn* for collaborative discussions with *JavaLearn.*	**3.33**
6	How would you rate the ease of using the *JavaLearn* interface for the activity and interactions?	**4.00**
7	How would you rate your overall satisfaction with *JavaLearn*?	**3.71**
8	How would you rate the success of *JavaLearn*? Success refers to whether you feel you learned more by using this tool than you would do without it.	**3.80**

The result shows that the learners gave favourable responses for *JavaLearn*. It has managed to provide the opportunity to the learners to persist in the problem solving activity.

Conclusions and Future work

This paper has presented an architecture for *JavaLearn* prototype which is able to engage the learners in a problem solving activity and the same time allows the students to "interact" with the system as part of the learning process activity. The system was built using rule-based agents. The agents acquired the intelligence through the finite state machine. The feedback from the students during the beta testing shows that the system had contributed to the enhancement of their and understanding on the subject matter. *JavaLearn* provides the following significance:

- It converge mixed-initiative interactions, web-based learning and collaborative learning in a single platform;
- It provides a computational model for the system based on the mixed-initiative interactions using finite machines to deliver learning to the learners;

- It enables the students to do an activity and at the same time engage in a “conversation” with the agent. Thus, it eliminates the human intervention; and
- The model proposed in this paper can be easily expanded to other subject areas.

Developing mixed-initiative systems that mimics the typical human-human interaction is a daunting task. The challenges of developing *JavaLearn* is that it must have a comprehensive knowledge bases (rules) and ‘predefined messages” as the agents are dependent on these knowledge bases in interacting with the learners. We are currently in the investigating the idea of using *JavaLearn* in mobile phones to support mobile learning. JADE provides a mechanism to develop such mobile applications.

References

Baker, M. & Lund, K. 1996. Flexibly structuring the interaction in a CSCL environment. *Proceedings of the European Conference on Artificial Intelligence in Education (EuroAIED ‘96),* 401-407.

Charles Rich and Candace L. Sidner, “COLLAGEN: A Collaboration Manager for Software Interface Agents”, *User Modeling and User-Adapted Interaction 8(3),* 1998, Kluwer Academic Publishers, pp 315-350.

Hanson, P., Sarah Judd and Charles Rich. 2009. Zeppelin Time: Exploring the Future of Mixed Initiative in Educational Games. http://web.cs.wpi.edu/~rich/courses/cs525u-s09/projects/ZeppelinTime/ProjectNotes.pdf

Huang, H. (2002). Toward constructivism for adult learners in online learning environments. *British Journal of Educational Technology, 33(1),* 2737.

Israel, Judith Lynne. 2003. *Collaborative learning enhanced by an intelligent support system.* Doctoral Dissertation. Temple University,

Menon S., Shakya J., Kumar V., Rule-Based Mixed-Initiative Scaffolding, International Workshop on Applications of Semantic Web Technologies for E-Learning (SW-EL), Banff, Canada, 2005.

Rao S., Kumar V., Hatala M., Gasevic D., Mixed-Initiative Interfaces to Recognize, Regulate, and Reflect Programming Styles, I2LOR Conference, November, Montreal, Canada, 2006.

Shakya J., Menon S., Doherty L., Jordanov M., Kumar V.S. Recognizing Opportunities for Mixed-Initiative Interactions based on the Principles of Self-Regulated Learning. Workshop on Mixed-initiative interactions for Problem-solving. AAAI Fall Symposia, Arlington, VA, 2005.

Thurmond, V., & Wambach,K. 2004. Understanding Interactions in Distance Education: A Review of the Literature. International Journal of Instructional Technology and Distance Learning.