A Prototype for an Intelligent Tutoring System for Students Learning to Program in Java™

Edward R. Sykes
School of Computing and Information Management, Sheridan College
1430 Trafalgar Road, Oakville, Ont., Canada, L6H 2L1
+1 (905) 845-9430 Ext. 2490
ed.sykes@sheridanc.on.ca

Franya Franek
Department of Computing and Software, Faculty of Science, McMaster University
1280 Main Street W., Hamilton, Ont., Canada, L8S 4L8
+1 (905) 525-9140 Ext. 23233
franek@mcmaster.ca

Abstract

The "Java™ Intelligent Tutoring System" (JITS) research project involves the development of a programming tutor designed for students in their first programming course in Java™ at the College and University level. This paper presents an overview of the architectural design including state-of-the-art web-based distributed architecture, the AI techniques used, and the programmer-optimized user interface. This project is a prototype being constructed which will model the domain of a small subset of the Java™ programming language in a very specific context. Research is in progress and it is hypothesized that the completed prototype will be sufficient to prove the concept and that a fully developed Java™ Intelligent Tutoring System will provide an interactively-rich learning environment for students that will result in increased achievement. Based on the success of similar Intelligent Tutoring Systems, it is also hypothesized that these students will be able to learn programming skills and knowledge more quickly and effectively than students in traditional educational settings.

Keywords: Intelligent Tutoring Systems, Web-Based Education, Programming Tutors, Artificial Intelligence in Education.

1. Introduction

Intelligent Tutoring Systems (ITS) are, in many respects, very similar to human tutors. Based on cognitive science and Artificial Intelligence (AI), ITS have proven their worth in multiple ways in multiple domains in Education [1, 2]. Currently, ITS can be found in core Mathematics, Physics, and Language courses in many schools across Canada, the United States, and various countries in Europe. ITS are growing in acceptance and popularity for reasons including: i) increased student performance, ii) deepened cognitive development, and iii) reduced time for the student to acquire skills and knowledge [1, 2, 3].

Intelligent Tutoring Systems that tutor and monitor programming have been developed and evaluated for many years in the field of Artificial Intelligence in Education. In many ways, programming has been a very productive domain in the evolution of most aspects of the field, including student modeling, knowledge representation, and the application of sound pedagogical principles. Effective programming requires a range of problem-solving and diagnostic strategies. The manner in which a student writes code provides rich insight into the reasoning processes of the student. As a result, programming provides an interesting domain for studying learning and cognitive processes.

The goal of the current research is to bring together recent developments in the fields of Intelligent Tutoring Systems, Cognitive Science, and AI to construct an effective intelligent tutor help students learn to program in Java™. In addition to contributing to understand the learning process in general, it is hoped that this research will have a positive impact on supporting instructors teaching Java™ programming in their institution. More than ever, this is an important area for institutions where there are more students wishing to learn to program, and where it is difficult to provide personalized instruction that they need [4]. Additionally, since there are a growing number of institutions investing in distance learning, this research will play a significant role to provide appropriate methods of teaching this key subject to students learning remotely.

2. Java ITS Model and Architecture

This section presents the model and architecture for the Java™ Intelligent Tutoring System. Four distinct components are presented that support JITS: the curriculum design, the AI module, the distributed web-based infrastructure, and the user interface design.

2.1. JITS Curriculum Design

This section describes the curriculum architectural model for JITS. Due to the complexity involved with semantic parsing, it is necessary to restrict the JITS to tutor a small subset of the Java programming language. The area of focus involves the following list of Java™ language basics:

a. variables (declaration, use, local vs. global),
b. operators, and
c. looping structures.

A database of records with the following structure will be constructed:

Given a Problem (P), there are n number of Solutions (S_1, S_2,…, S_n), with a number (m) of categories of classifiable incorrect responses (R_1, R_2,…, R_m). For each incorrect response category there is a finite number (t) of appropriate hints (H_1, H_2,…,H_t). Table 1 illustrates an example of a problem with a solution and some incorrect responses.

Table 1. Java™ ITS Curriculum Architecture

Problem:
Write a program called "Summer" which adds all the integer numbers from 1 to a specified number (N). For example, if N were assigned the value 10, then the sum of the numbers from 1 to 10 is 55.

Program specifications:
This program requires the use of a for-loop structure. A skeleton structure of the solution is given. Fill in the code to complete this program.

```
OUTPUT>Sum = 55
```

Skeleton Program (located in Source Code area):

```
public class Summer {
 public static void main(String[] args)
 {
   int sum = 0;
   int i = 0;

   /* student writes code here */

   System.out.println("Sum = " + sum);
 }
}
```

Solution (one of n):

```
public class Summer {
   public static void main(String[] args) {
     int sum = 0;
     int i = 0;
     for (i = 1; i <= 10; i++) {
         sum += i;
     }
     System.out.println("Sum = " + sum);
  }
}
```

Incorrect response #1 (student response area):
(redeclaration of variable 'i')

```
     for (int i = 1; i <= 10; i++) {
         sum += i;
     }
```

Incorrect response #2:
(sum is 0, as the body of the loop is never executed)

```
     for (i = 1; i > 10; i++) {
         sum += i;
     }
```

Incorrect response #3:
(adding 1 instead of variable 'i': results in sum being lower than expected)

```
     for (i = 1; i <= 10; i++) {
         sum += 1;
     }
```

Incorrect response #4:
(sum does not include last integer, i.e., '10')

```
     for (i = 1; i < 10; i++) {
         sum += i;
     }
```

etc.

The astute reader recognizes that there are limitless possibilities for student responses and the system cannot simply list incorrect responses coupled with feedback messages. For instance, the student could write:

```
sum = (n+1)*(n/2);   or
sum = (n*n+n)/2;
```

Both answers are completely correct and the system needs to recognize these types of responses and not merely respond back to the student indicating a failure. Testing the correctness of a program is not an easy task, and cannot be achieved just by giving a set of fixed responses. JITS is designed to be pedagogically sound. So, although the above formulas result in correct answers, this is not the final goal of the tutoring system. Rather, JITS focuses on the methodology by which a student attempts to solve a problem. Conventions, style, and professional programming techniques are modeled in JITS. In this fashion effective tutoring may take place. These pedagogical issues as they are designed in JITS are addressed in the following sections.

2.2. JITS AI Module

In order for JITS to provide intelligent feedback to the student the AI module relies on a collection of information: the problem statement, the problem specification, student's code, the established student model, the expert model, the Java™ Parser, the syntactic_decision_tree, the semantic_decision_tree, the Java™ Parse Tree, the output from the Java™ compiler, and the result from the Java™ runtime engine. Obviously, based on the context, some of this information will not be available. The two decision trees (i.e., syntactic_decision_tree, and semantic_decision_tree) represent the strategic and judgmental knowledge for the specific programming problem currently being examined by the student [5].

The information gathered by the AI module is carefully scrutinized so that appropriate hints can be generated for the student. For instance, if the Java™ Parser fails, and there was a solution provided (i.e., *s*2), the "Fuzzy Scanner" (FS) module computes the edit distance between the student's code (*s*1) and the solution (*s*2), and constructs a transformation function string (i.e., $T : s1 \rightarrow s2$). T involves all insertions, deletions, transpositions, and character changes that are required to transform *s*1 into *s*2. In the case where no solution has been provided, as will be the case with complex programming problems (since many solutions are possible), the FS attempts to transform *s*1 into a program that is syntactically well-formed. The AI module then goes beyond the student and constructs feasibly-sound variations of the student's code and proceeds to compile and run these. The information produced by the FS module is fed into the syntactic_decision_tree to determine appropriate feedback.

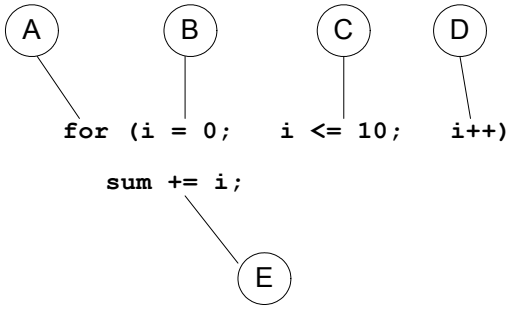

Figure 1. Sub-sections of expert model solution

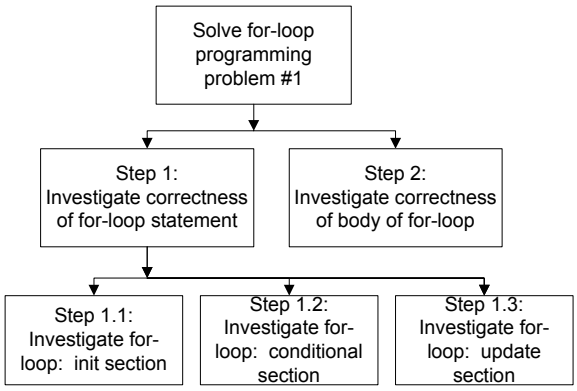

Figure 2. High-level functional decomposition tree

Using the arithmetic sum problem described in Table 1, Figure 1 depicts how the expert model's solution may be divided into discrete sections. Based on this, Figure 2 represents the high level functional decomposition tree for this problem with Figure 3 presenting a high-level flow chart showing the process.

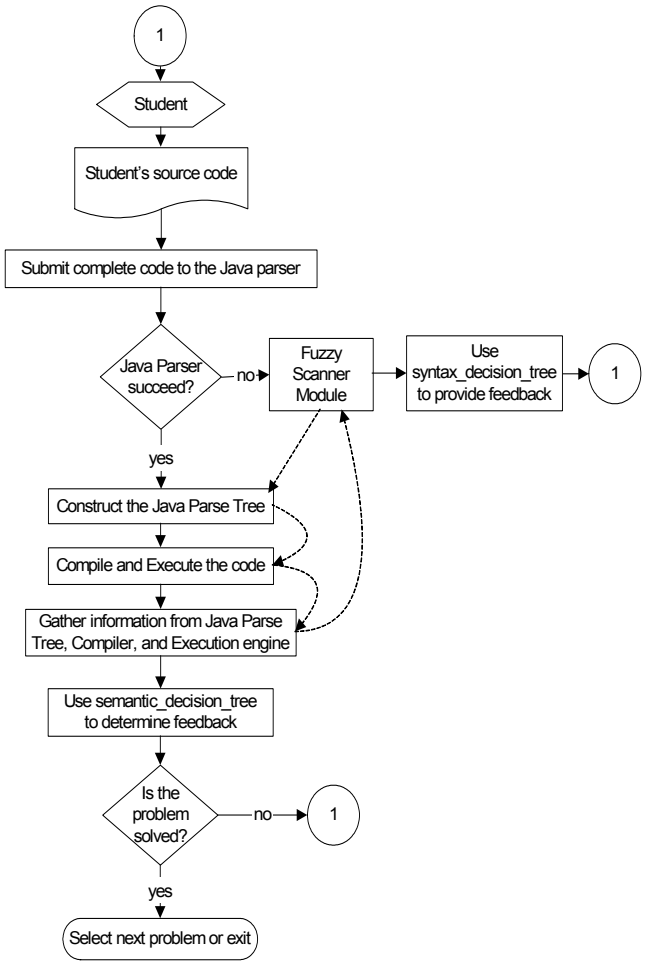

Figure 3. Flow chart of JITS AI Module

2.2.1. JITS AI Module: Feedback

The information gathered by the AI module previously described is then carefully scrutinized so that appropriate hints can be generated for the student. An expert system (supporting the decision trees), and two methods support the feedback mechanism: `general_hint(int context, String snippet)` and `specific_hint(int context, String snippet)`. Although the two decision trees isolate the

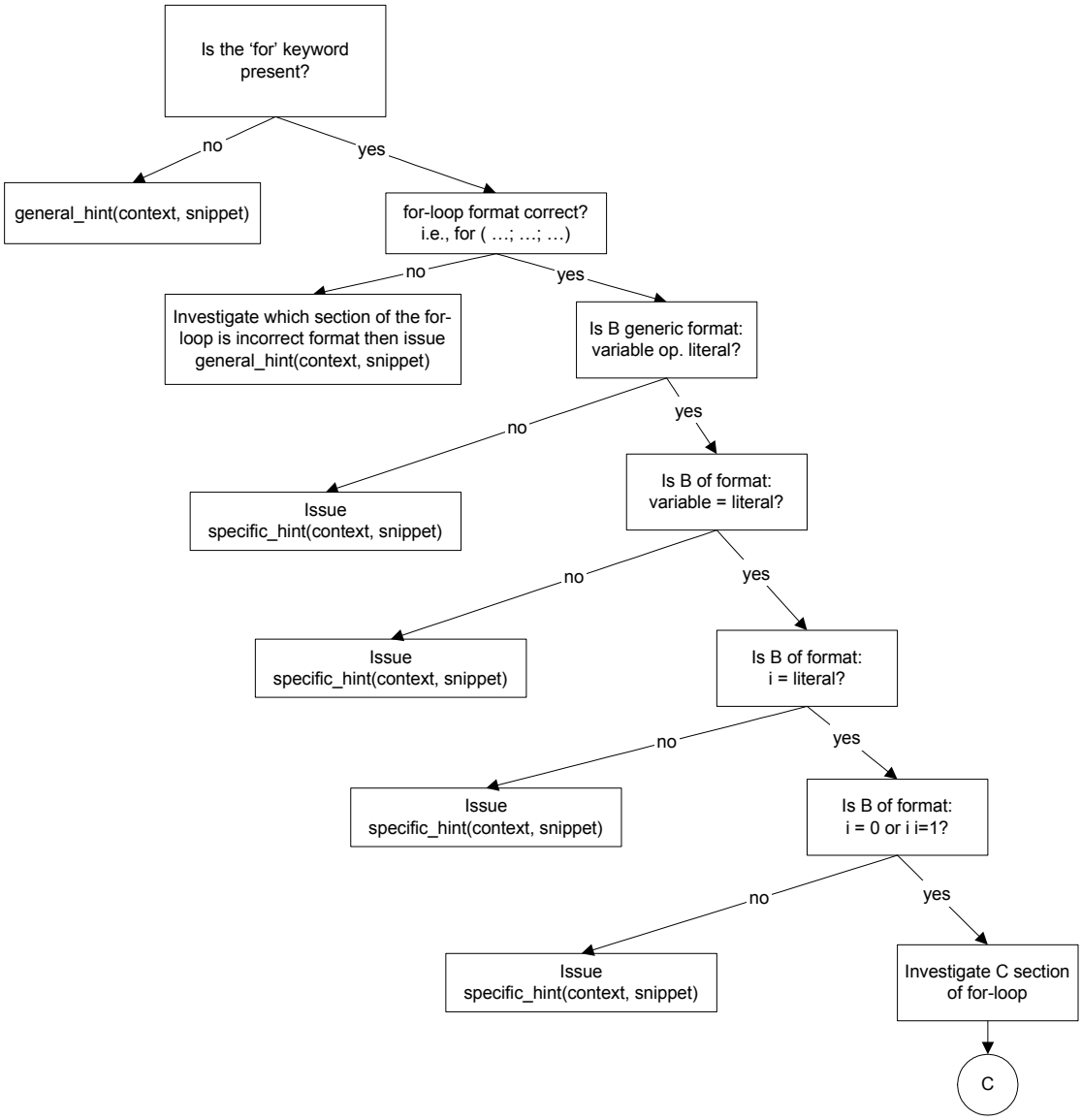

Figure 4. JITS semantic_decision_tree (sections A and B from Figure 1 only)

specific area of error within the student's code, additional fine-grained analysis occurs within these methods. These methods are passed an integer representing the context in which the current programming issue has been identified. The second argument, *`snippet`*, represents the small portion of code associated with the given *`context`*. For example, if the student submitted: `sum = (n*n+n)/2;` only, then `general_hint()` would issue a message such as, *"Your answer is correct. However, the program specification is asking for a solution using a for-loop construct. Please revise your code."* Specific hints are code-specific generated in the same fashion as a human tutor would during troubleshooting to pinpoint the exact situation in which a syntactic or logic error has occurred. Figure 4 depicts a small section of the semantic_decision_tree (please refer to sections A and B from Figure 1).

2.3. JITS Distributed Web-based Infrastructure

The JITS infrastructure supports the student via a browser accessing information from the tutor via an HTTP request/response process model. The processing is accomplished by Enterprise JavaBeans™ within a J2EE compliant server in combination with a web server supporting the presentation logic for the tutor. The presentation layer uses JavaServer Pages™ technology which communicates with the home interface of the bean for processing and returns a simple page back to the student's browser (e.g., html, xml, etc.). During processing the bean gathers all the information about the student's code and submits it to the AI module for processing. The infrastructure architecture uses a JDBC connection from the Enterprise JavaBeans™ to an external database which stores and retrieves specific

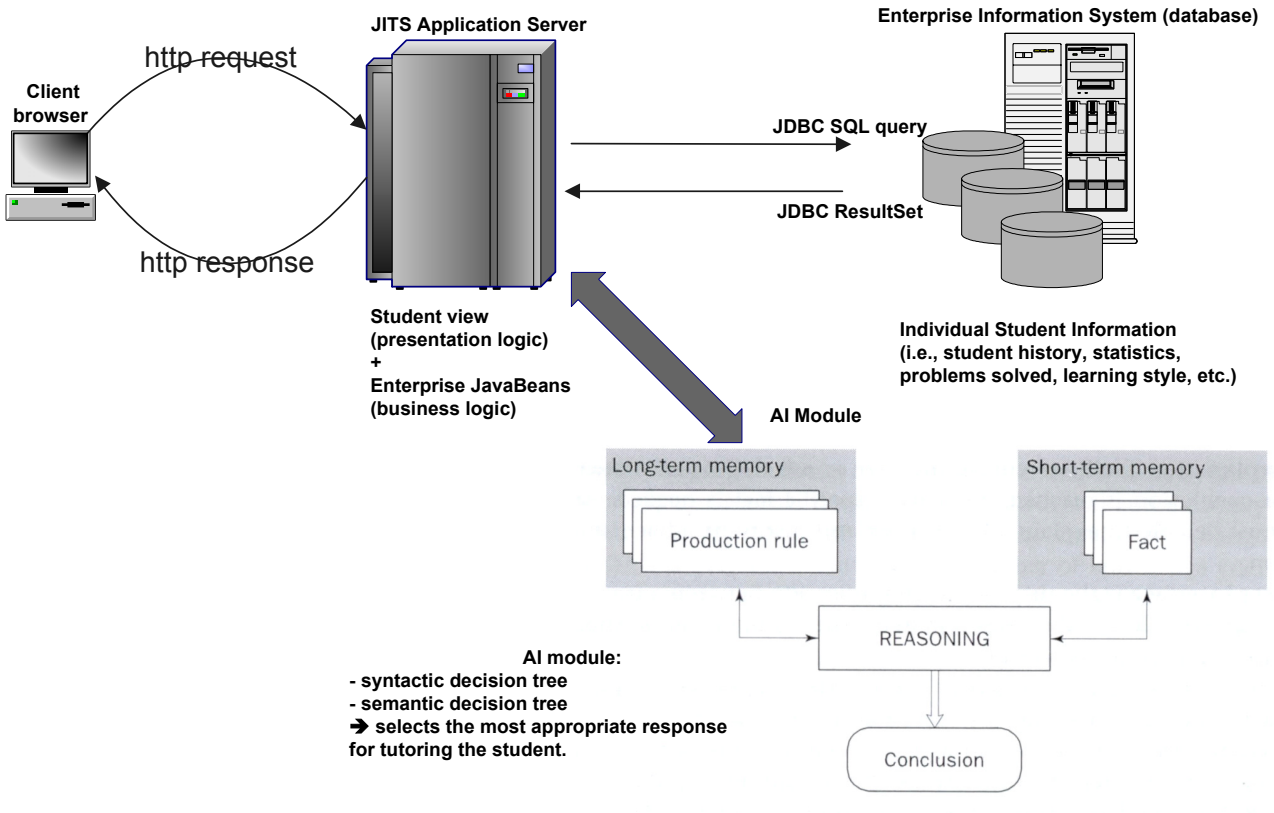

Figure 5. JITS Distributed Web-based Infrastructure

information about the student including student history and performance statistics.

The proposed architecture has numerous benefits. It is scalable, platform-independent, and lightweight. The student will never need to install software on their machine and will not need a high-speed network connection to use JITS. Other benefits include fast execution as all processing is done on the J2EE server and the middle-tier web server which typically have much faster and more efficient hardware than typical PCs. The net result is a product that increases the accessibility for JITS to many students – a vital requirement for an equitable and successful educational product in today's Internet-ready community. Figure 5 presents a pictorial view of the JITS Distributed Web-based Infrastructure.

2.4. JITS User Interface

The interface for computer-based programming tutors is a significant factor that was given careful consideration during the design of JITS. The user interface is based on a presentation format implemented in many popular Integrated Development Environments used by professional programmers (e.g., Visual Café, JDeveloper).

Upon connecting to JITS website, the student's browser displays the working environment for JITS. An appropriate skill-level problem is selected or the problem that last attempted is presented to the student.

The student types in a solution in the Source Code Area and presses 'Parse'. This invokes a call to the corresponding Enterprise JavaBean™ representing the student. Information is gathered (i.e., student model, Java™ Parser, compiler, runtime engine, etc.) and submitted to the AI module.

If the parser does not succeed then the AI module will determine the appropriate response based on the syntactic_decision_tree. Otherwise, JITS goes beyond the student and attempts to compile and execute the code. This yields additional information for the AI module to construct intelligent feedback. This information is used by the semantic_decision_tree. Specific feedback is generated and sent to the student's browser.

The student, at any time, may explicitly request a hint from JITS, view the solution, opt to quit the problem and select another, and view their performance history based on statistics including problems attempted, problems solved, number of attempts on a problem, and problem difficulty. The JITS user interface is shown in Figure 6.

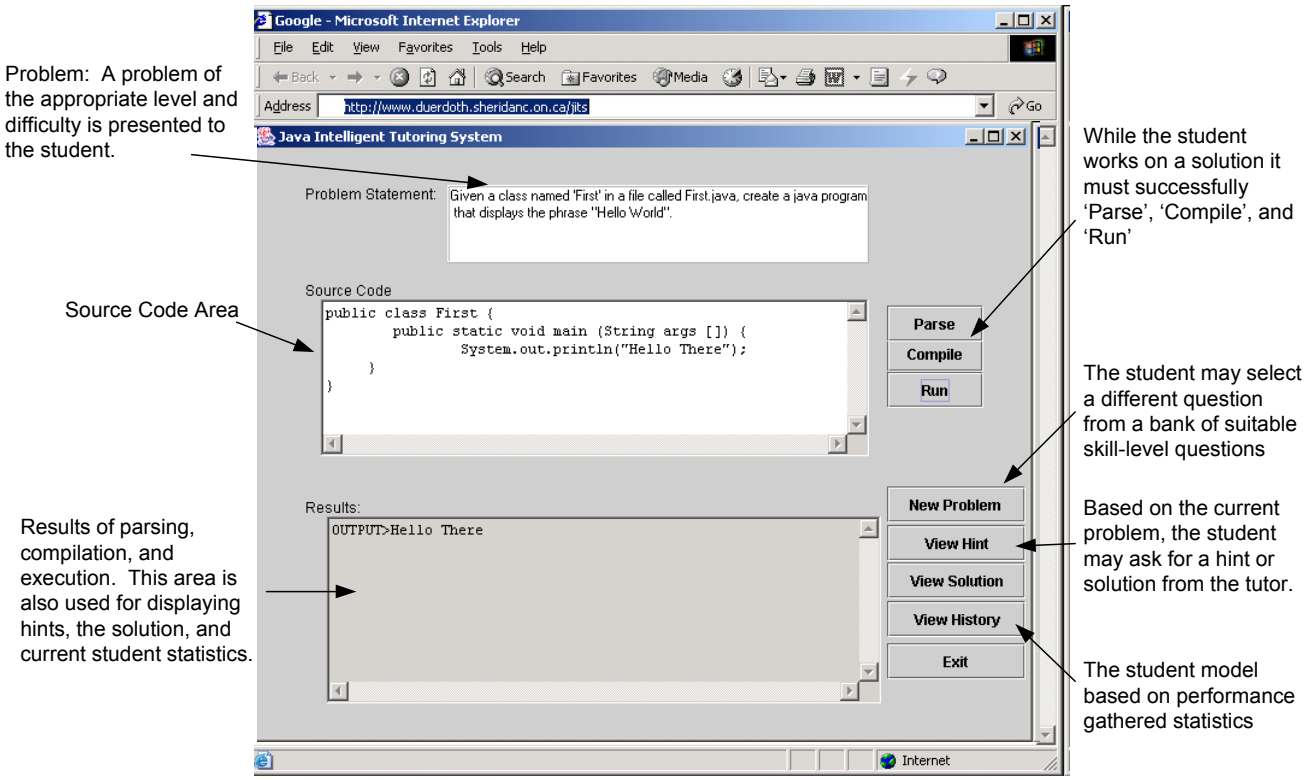

Figure 6. JITS User Interface

3. Conclusions

In summary, the Java™ Intelligent Tutoring System prototype is designed using advanced cognitive science and AI techniques promoting the necessity for on-going research and development in the field of web-based educational tools. This research project is significant since it has the potential to be applied to numerous programming courses at the College and University level. Furthermore, it is important to the field of Education in both e-learning and traditional settings. The project in progress is based on sound theories and practices used in successful Intelligent Tutoring Systems and draws from the achievements ITS researchers have had in related projects.

4. References

[1] Anderson, J. R., Corbett, A. T., Koedinger, K. R., & Pelletier, R. (1995). Cognitive Tutors: Lessons learned. *The Journal of the Learning Sciences, 4,* 167-207.

[2] Woolf, B., P., Beck, J., Eliot, C., & Stern, M. (2001). Growth and maturity of intelligent tutoring systems: A status report, In K. D. Forbus & P. J. Feltovich (Eds.), *Smart machines in education* (pp. 100-144). Cambridge, MA: MIT Press

[3] Graesser A. C., Person, N. K., & Harter, D. (2001). Teaching tactics and dialog in autotutor. *International Journal of Artificial Intelligence in Education, 12,* 12-23.

[4] Koedinger, K. R. (2001). Cognitive tutors. In K. D. Forbus & P. J. Feltovich (Eds.), *Smart machines in education* (pp. 145-167). Cambridge, MA: MIT Press.

[5] Scott, A. C., Clayton, J., E., & Gibson, E., L. (1991). *A Practical Guide to Knowledge Acquisition*. Menlo Park, CA: Addison-Wesley.