A PROTOTYPE FOR AN INTELLIGENT TUTORING SYSTEM FOR STUDENTS LEARNING TO PROGRAM IN JAVA™

E. R. Sykes* and F. Franek**

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 or 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 gain knowledge more quickly and effectively than students in traditional educational settings.

Key Words

Web-Based Education, Programming Tutors, e-Learning, Intelligent Tutoring Systems

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 this 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 understanding 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.

* School of Applied Computing and Engineering Sciences, Sheridan Institute of Technology and Advanced Learning, 1430 Trafalgar Road, Oakville, Ont., Canada, L6H 2L1; e-mail: ed.sykes@sheridanc.on.ca

** Department of Computing and Software, Faculty of Science, McMaster University, 1280 Main Street W., Hamilton, Ont., Canada, L8S 4L8; e-mail: franek@mcmaster.ca

(paper no. 202-1454)

2. Java ITS Model and Architecture

This section presents the model and architecture for the Java™ Intelligent Tutoring System. JITS is designed with two distinct mechanisms of functionality:

2.1. 'A'-Type JITS Functionality

In many Intelligent Tutoring Systems, the process of authoring involves a professor to provide a set of problems, their specifications, and corresponding solutions. In JITS, this type of functionality is provided for very straight-forward programming problems.

The 'A'-type functionality is solved by a Dynamic Programming Algorithm (DPA) edit-distance algorithm. This topic is discussed in detail in section 6.1.

2.2. 'B'-Type JITS Functionality

The second mechanism of functionality that JITS provides is a consequence of the limitations from 'A'-type functionality described above. In many programming problems there are often many solutions. A professor may provide one solution to a problem but there may be many other solutions that are equally as suitable. As a result, the most reasonable approach is to request the professor to author only the problem, the problem specification, and the output (i.e., desired results); JITS needs to determine the rest.

The 'B'-type functionality requires much more rigor in terms of attempting to ascertain the 'intent' of the student by analyzing the code. The difficulty in these types of problems is that there is no coded solution from which JITS can use as a comparison. As a result, a specialized intent recognition scanner-parser algorithm prototype has been developed as a means of determining the intent behind the student's submission. This algorithm is described in greater detail in section 6.2.

3. JITS Overview and Framework

The following sections describe the JITS framework from a high-level perspective. Fig. 1 presents a flowchart of how JITS processes the student's submission. Four distinct components are presented that support JITS: the curriculum design, the AI module, the distributed web-based infrastructure, and the user interface design.

4. 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 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.

Table 1 illustrates an example of a problem with a solution and some incorrect responses.

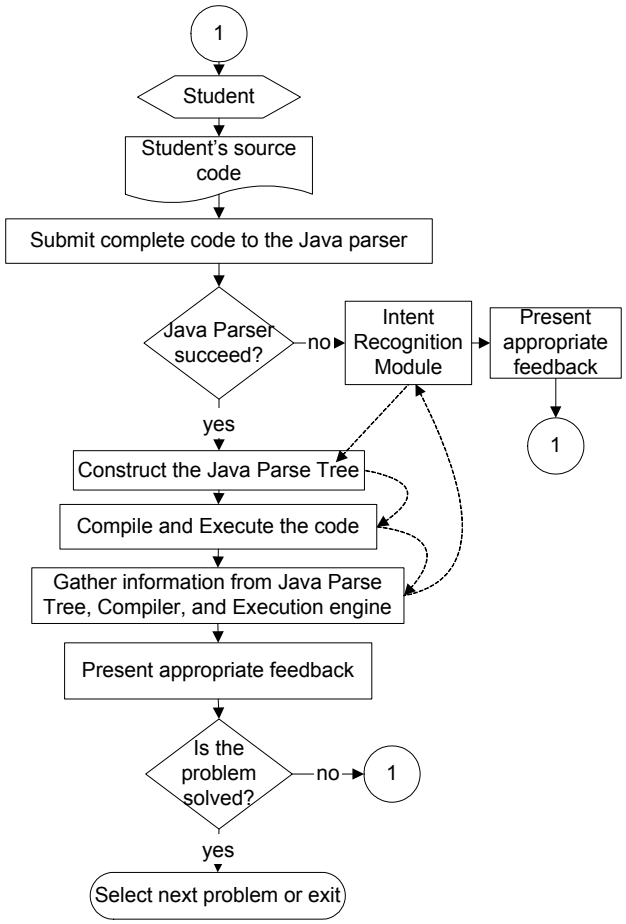


Figure 1. Flow chart of JITS process model

Table 1
Java™ ITS Example Problem

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
```

table continued…

Skeleton Program (located in Source Code area):

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

   /* student writes code here */

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

Solution (authored by Professor):

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

Incorrect response #1 (student response area):
(no declaration of variable 'i')

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

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

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

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

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

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

```
     for (int 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 [5]. 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 [6]. Just as presented in this example, JITS is focusing the student on the problem on hand by specifying the location where code may only be written (i.e., the "`/* student writes code here */`"section). Conventions, style, and professional programming techniques are modeled in JITS [6]. In this fashion effective tutoring may take place. These pedagogical issues as they are designed in JITS are addressed in the following sections.

5. 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 Java™ Parse Tree, the output from the Java™ compiler, and the result from the Java™ runtime engine. Based on the context some of this information will not be available. However, the goal of this module is to carefully scrutinize all available information so that appropriate feedback may be generated for the student. This is accomplished by the core component of the AI module – the Intent Recognition module.

6. JITS Intent Recognition Module

The purpose of the Intent Recognition (IR) module is to ascertain the most probable submission of code the student intended. As identified in fig. 1, the IR is invoked when the standard Java™ parser fails. The IR first determines whether there is a solution available for the current problem. If so, then the IR will employ the 'A'-Type JITS functionality, otherwise 'B'-Type functionality will be used.

6.1. Inside the IR Module: 'A'-Type Functionality

In this context, the IR module has an existing solution for the given problem. Let the student's submission be *s1*, and the solution *s2*. The IR module performs pattern matching between the two strings to generate a transformation string and to calculate the edit distance

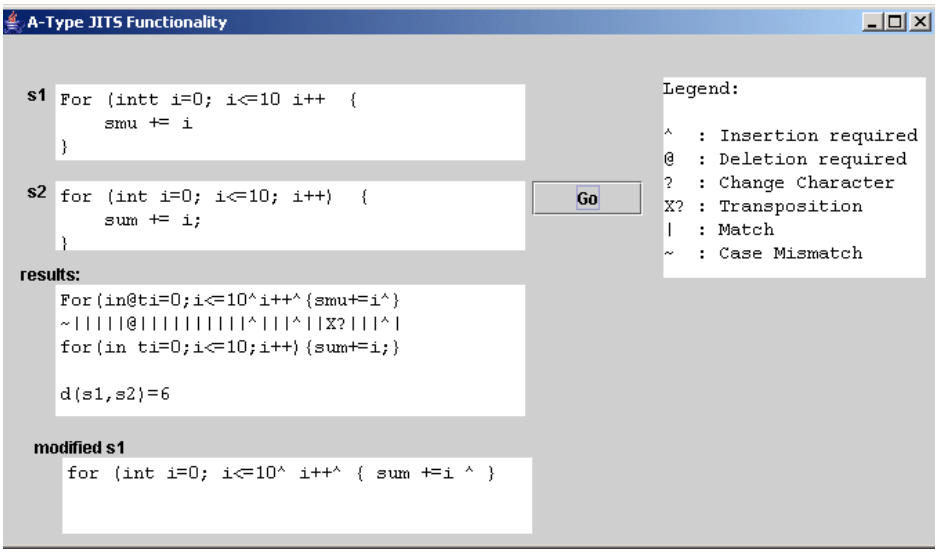


Figure 2. A-Type JITS Functionality

between the student's code and the solution. The IR 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 $s1$ into $s2$. The IR then constructs feasibly-sound variations (ie., *modified s1*) of the student's code and proceeds to compile and run these. Continuing using the example in table 1, consider the student's submission (*s1*):

```
For (intt i = 1; i <= 10  i++ {
    smu += i
}
```

and the desired solution string (*s2*):

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

As shown in fig. 2, *s1*, *s2*, and *modified s1* depict the pattern matching procedure. The *results* region shows the transformation function indicating the insertions, deletions, transpositions, and character changes using symbols described in the *Legend*. The IR module passes this information on to the AI module to determine the appropriate response to the student. This is necessary as the AI module has additional information such as the student's model, and the expert model, to conduct effective tutoring [5, 6].

6.1.1. *Pattern Matching Algorithm*

The pattern matching algorithm employed in the 'A'-type JITS functionality is a DPA that generates the transformation function (i.e., T), and calculates the edit distance. The cost for an insertion, deletion, transposition, or character change is 1.

6.1.2. *'A'-Type Tutoring Process*

Before the IR module makes the changes as indicated in the transformation function, the student is asked specific questions for intent clarification purposes. In this fashion, only modifications that the student knows upfront about will be made to the code.

So, using example 1 in table 1, the following dialogue between JITS and the student would arise:

```
For (intt i = 1; i <= 10 i++ {
    smu += i
}
```

JITS: "I think you meant the keyword 'for'. Is this correct?"
Student: Yes
JITS continues processing presenting the modified code below…

```
for (intt i = 1; i <= 10 i++ {
    smu += i
}
```

JITS: "I see 'intt'. Do you mean the keyword 'int' ?"
Student: Yes
Resulting code:

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

JITS: "Did you mean to insert a ';' between the '10' and 'i++'?"
Student: Yes
Resulting code:

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

JITS: "There seems to be something missing. I think we are missing a ')' between 'i++' and '{'. Do you agree?"
Student: Yes
Resulting code:

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

JITS: "I found 'smu' but there is a very similar identifier in this program named 'sum'. May I rename 'smu' to 'sum'?"
Student: Yes
Resulting code:

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

JITS: "I think we're missing a ';' at the end of this line. Do you agree?"
Student: Yes
Resulting code:

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

The tutoring process is dynamic. At any time the student is able to interject, disagree with JITS' suggestions, and modify the source code [7]. This results in JITS having to reiterate the IR algorithm and the tutoring process begins again.

6.2. Inside the IR Module: 'B'-Type Functionality

The IR's responsibility is to systematically employ a minimum distance error-correcting scanner-parser algorithm with the goal of remedying the student's code [8, 9]. The IR algorithm is explained in the following section.

6.2.1. Syntax Error Correction Strategy

Let L be a nonempty set of strings over the finite set of symbols used in the Java programming language (i.e., Σ). It is assumed that a string not in L may be derived from some sentence in L by a sequence of error-transformations. The IR module recognizes four types of syntax errors:

i) the replacement of a symbol by another symbol,
ii) the insertion of an extraneous symbol,
iii) the deletion of a symbol, and
iv) the transposition of two adjacent symbols.

These four errors can be represented by four transformations T_R , T_I , T_D , and T_S from Σ' to the subsets of Σ' defined as follows. For x and y in Σ':

i) xby is in $T_R(xay)$ for all $a \neq b$
ii) xay is in $T_I(xy)$ for all $a \in \Sigma$
iii) xy is in $T_D(xay)$ for all $a \in \Sigma$
iv) xy is in $T_S(yx)$

The goal of the IR is to select a sequence of intermediate strings and error transformations such that the result is a transformation sequence that produces an acceptable token for the parser.

For example, suppose $L=\{cde\}$. Given a string *ddfe,* the first *'d'* is a replacement error and the *'f'* is an insertion error because:

$$cde \xrightarrow[T_R]{} dde \xrightarrow[T_I]{} ddfe$$

Using Java™ for another example, consider the following declaration:

```
publik status flot TAX=5;
```

The IR would construct the following Transformation Sequence:

$publik \xrightarrow[T_R]{} public$

$status \xrightarrow[T_R]{} statis \xrightarrow[T_R]{} static$

$flot \xrightarrow[T_I]{} float$, resulting in the correct syntax:

```
public static float TAX=5;
```

6.2.2. IR Scanner-Parser Algorithm

This section describes the Intent Recognition Scanner-Parser Algorithm. The grammar that the scanner and parser operate under is the most current version of the J2SE – Sun Microsystem's Java 1.4.2_02 specification.

The algorithm is presented as follows:

i) The scanner examines the student's code and attempts to extract a token. Let S be the stream of characters to be validated as a token.
ii) The validation process ensues in which comparisons are done using the reserved words and keywords of Java (Table 2), and the symbol table (Table 3).

iii) If the scanner cannot ascertain an appropriate token then the transformations T_R , T_I , T_D , and T_S are employed in an attempt to convert S into a valid token (i.e., a reserved word, a keyword, or a new identifier).

iv) This Transformation Sequence (TS) is recorded by the scanner in a special table called the Transformation Sequence Table (TST).

v) After a sufficient number of transformations (i.e., k-error corrections), a token will be constructed.

vi) The token is submitted to the parser.

vii) The parser asks the question: "In the current context, has a reasonable token been accepted?"

if (true) then

parser 'locks onto' this token by adding it to the current parse tree.

else

reject the current form of the token and communicate this back to the scanner so that the scanner can make appropriate modifications to the transformation sequence, or construct an appropriate token based on the context. For example, ';', indicating the end of a statement may need to be created to meet the needs of the parser to complete the parse tree.

viii) Repeat i) through vii) until all input from the student's source code has been processed, and the parser has completed the construction of the parse tree.

Table 2
Java™ Reserved Words and Keywords

abstract	else	interface	super
boolean	extends	long	switch
break	false ***	native	synchronized
byte	final	new	this
case	finally	null ***	throw
catch	float	package	throws
char	for	private	transient
class	goto *	protected	true ***
const *	if	public	try
continue	implements	return	void
default	import	short	volatile
do	instanceof	static	while
double	int	strictfp **	

Note:

* indicates a keyword that is not currently used

** indicates a keyword that was added for Java 2

*** `true`, `false`, and `null` are reserved words.

Table 3
Symbol table

Lexeme	**Token**	**Type** (identifier, method_name, reserved_word, or keyword)	**Attribute Values**
int	INT	keyword	
for	FOR	keyword	
foobar	IDENTIFIER	method_name	
sum	IDENTIFIER	identifier	value: 1
true	TRUE	reserved_word	
=	ASSIGNMENT		
…	…	…	…

6.2.3. IR Scanner-Parser Example 1: Forward Processing

Based on the problem described in Table 1, the following example describes how the IR scanner-parser algorithm operates. Suppose JITS did not have available the solution as presented in Table 1. Also consider a student's code submission as follows:

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

Based on this scenario, JITS would employ the IR scanner-parser algorithm. The first string of characters are extracted as 'For'. The search commences through the keywords, reserved words, and symbol table for an exact string match. This having failed, pattern matching ensues, by employing the transformation functions (T_R, T_I, T_D, and T_S). The string 'For' in the input would be converted to the keyword 'for', and in the scanner's Transformation Sequence Table would reside the details of T_R. This token would be passed to the parser which would 'lock onto' it. The scanner then reads the next symbol (i.e., '(' left-parenthesis) and passes it to the parser, which in turn attaches it to the current parse tree. The string 'intt' is read next which undergoes the same treatment that the 'For' encountered. However, instead of a T_R transformation, a T_D would be recorded for the Transformation Sequence. Step by step, the scanner scrutinizes the input strings and attempts to classify each into a recognizable token for the parser. Fig. 3 presents a pictorial view of the procedure.

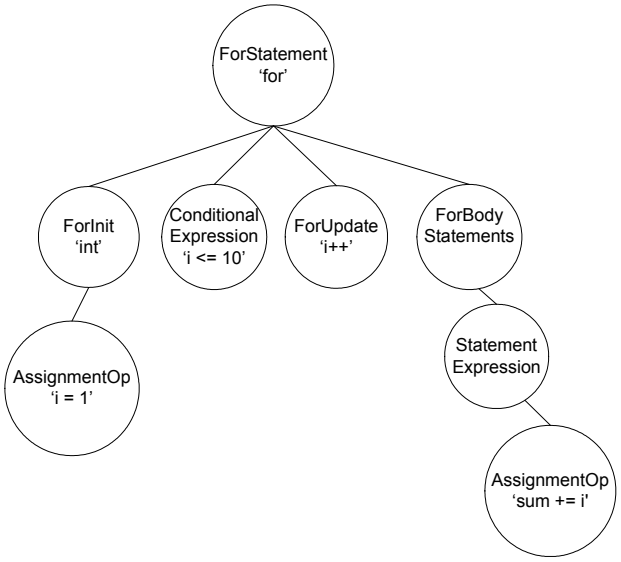


Figure 3. IR Scanner-Parser – parse tree construction

6.2.4. IR Scanner-Parser Example 2 – Forward and Backward Processing

In the previous example no situation arose where the parser rejected the token submitted by the scanner. Realistically, there will be times when the parser cannot accept the token delivered by the scanner. Consider the following:

```
For (intt i = 1; i <= 10  i++ {
     smu += i
}
```

The first string of characters would be transformed in the same manner as in example 1. However, after the scanner successfully identifies token '10' as an integer, the scanner would happily submit the next token, identifier 'i'. It is here the parser would reject to continue. The last token, 'i', does not fit the grammar for the for-statement (i.e., for (initialization ; test ; increment)). The parser would ask the scanner to create, or revise the Transformation Sequence to satisfy the parser's needs to complete the sentence according to the grammar. In this example, several tokens representing symbols would be created by the scanner representing ';', ')', and ';' for the three remaining syntax errors respectively.

6.2.5. 'B'-Type Tutoring Process

The underlying mechanism for analyzing the student's submission is quite different between 'A'-type and 'B'-type intent recognition. However, the tutoring process is virtually the same. In the 'B'-type mode, before the parser 'locks-on' to a token from the scanner, the IR module communicates the proposed changes with the student before actually making the change. This dialogue between the student and JITS would be the same as discussed in section 6.1.2.

6.3. Logic Errors

The IR module and tutoring processes previously described do not address issues associated with logic errors. So, even though the IR algorithm and tutoring process will result in a source program that will compile, there is no guarantee that it will satisfy the program requirements.

Once the IR module has completed the modification of the submitted code to one that parses, JITS uses information from the program specifications, the student model, the expert model, and the Java™ run-time engine to extract more information regarding the correctness of the student's program.

7. 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 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. Fig. 5 presents a pictorial view of the JITS Distributed Web-based Infrastructure.

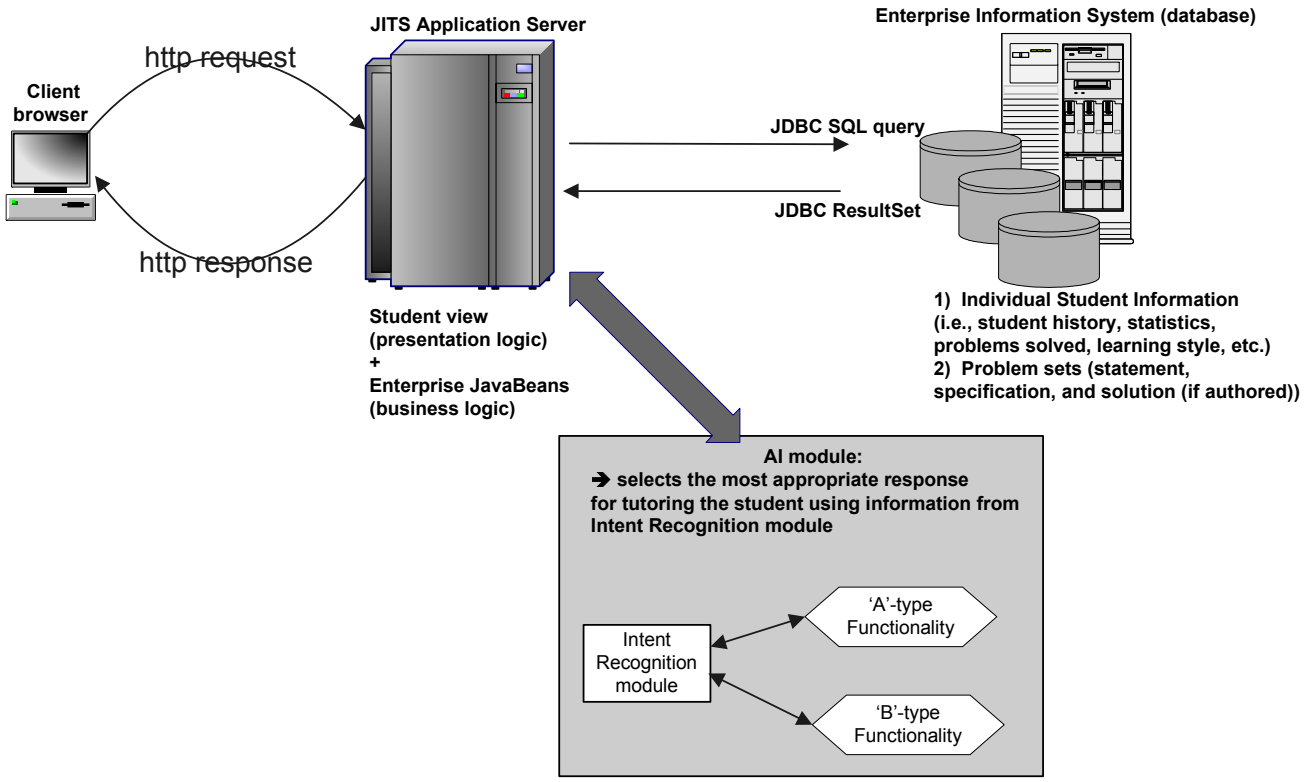


Figure 5. JITS Distributed Web-based Infrastructure

8. 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 diagnosis of the IR module. 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 so that 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 fig. 6.

9. Efficiency Considerations

The algorithms employed in this Intelligent Tutoring System are quite demanding in terms of the time required as a function of the size of input. The Intent Recognition module with the 'A'-type and 'B'-type functionality are the two computationally expensive areas in JITS. For instance, the 'A'-type pattern recognition algorithm uses a recursive decent procedure which is requires time proportional to the square of the length of input. The 'B'-type algorithm in the IR module requires time proportional to the cube of the length of input [9]. That is, $O(N^3)$, where N is the number of characters in the source program. Clearly, these are not efficient algorithms. However, JITS is not intended for programs of any size greater than 50 lines of code. As a result, considering such small values of N, the time cost would not be even noticeable to students. The purpose of JITS is to tutor beginning programming students at the College and University level and not to compile several hundred thousand lines of source code.

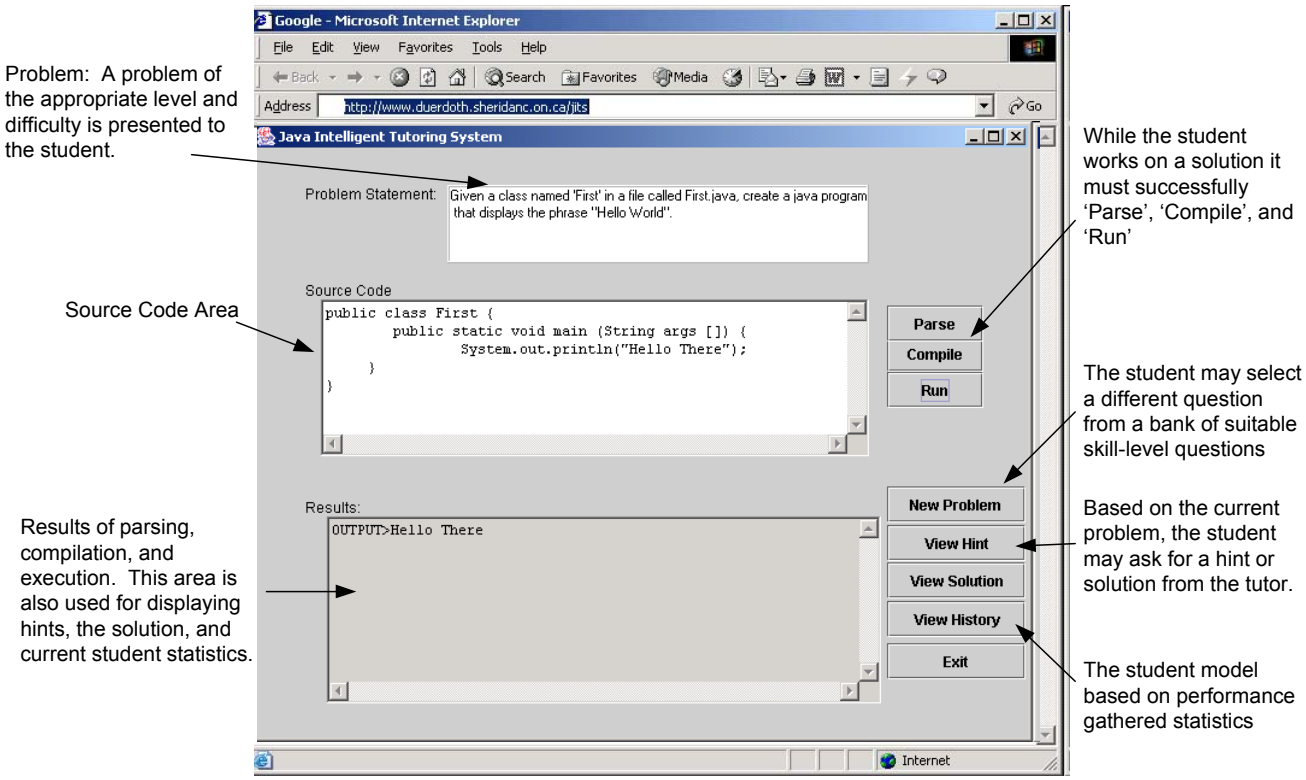


Figure 6. JITS User Interface

10. Conclusions

In summary, this research paper presented recent developments related to the Java™ Intelligent Tutoring System Prototype. The Intent Recognition module, comprised of the 'A'-type DPA edit-distance pattern matching algorithm and the 'B'-type IR scanner-parser, is based on sound theories, pattern recognition techniques, and error-correction strategies. The ultimate goal of the Intent Recognition module in JITS is to understand the 'intent' of the student by carefully analyzing the student's code and to communicate this to the AI module to effectively tutor the student through programming problems.

This research is significant since it has the potential to be applied to many programming courses at the College and University level. This research is also quite timely considering the tremendous growth of web-based educational tools, and that Java™ has become an extremely popular programming language everywhere in the world.

Acknowledgements

Edward R. Sykes would like to thank his wife, Michele for her tireless efforts in reviewing his papers and helping him in his research. Dr. Franek would like to thank Edward Sykes for his enthusiasm and energy with which he is undertaking his graduate studies.

References

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

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

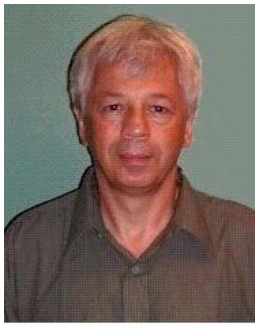
[3] A. C. Graesser, N. K. Person, Teaching tactics and dialog in autotutor, *International Journal of Artificial Intelligence in Education, 12,* 2001, 12-23.

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

[5] E6R. Sykes, Java™ intelligent tutoring system model and architecture: *Proceedings of American Association of Artificial Intelligence Spring Symposium on Human Interaction with Autonomous Systems in Complex Environments,* Menlo Park, CA, 2003, 187-193.

[6] E. R. Sykes & F. Franek, A prototype for an intelligent tutoring system for students learning to program in Java: *Proceedings of the IASTED International Conference on Computers and Advanced Technology in Education,* Rhodes, Greece, 2003, 78-83.
[7] A. C. Scott, J. E. Clayton, & E. L. Gibson, *A Practical Guide to Knowledge Acquisition* (Menlo Park, CA: Addison-Wesley, 1991).
[8] A. V. Aho, R. Sethi, J. D. Ullman, *Compilers: principles, techniques, and tools* (Menlo Park, CA: Addison-Wesley, 1988).
[9] A. V. Aho & T. G. Peterson, A minimum distance error-correction parser for context-free languages, *SIAM Journal of Computing, 1,* 1972, 305-312.

Biographies



Frantisek Franek is a full professor of Computer Science and Mathematics at McMaster University, Hamilton, Ontario, Canada. He obtained a Ph.D. in pure mathematics at University of Toronto, Toronto, Ontario, Canada, and a Doctorate in Natural Sciences with specialization in Computer Science at Charles University of Prague in his native Czech Republic. His research interests span set theory to combinatorics to computer science. Prof. Franek has published over 60 papers and has given over 80 presentations on these topics. His book "Memory as a programming concept in C and C++" is to be released by Cambridge University Press early next year. In Computer Science he specializes in algorithms on strings, however he always maintained a strong interest in AI and AI techniques. Besides that his interests include compilers and compiler techniques, and databases and their applications.

Edward R. Sykes is currently a Professor of Computer Science and Program Coordinator of the Computer Science Technology program in the School of Applied Computing and Engineering Sciences at the Sheridan Institute of Technology and Advanced Learning. Currently, Mr. Sykes is working on his Ph.D. developing a Java Intelligent Tutoring System and conducting research as to its effectiveness. Since 1994, Mr. Sykes has taught numerous courses including programming (e.g., Visual Basic, C, C++, Perl, and Java), database (e.g., SQL, PL/SQL, administration, backup and recovery, tuning), and operating systems. He is also very involved in the Research department at Sheridan. Currently, he is Sheridan's principal investigator for the Centre for Contemporary Canadian Art project (http://www.ccca.ca/).