VersaTutor – Architecture for a Constraint-Based Intelligent Tutor Generator

Viswanathan Kodaganallur
Seton Hall University
400 South Orange Avenue
South Orange, NJ, USA
(1) 973-761-9716
kodagavi@shu.edu

Rob R. Weitz
Seton Hall University
400 South Orange Avenue
South Orange, NJ, USA
(1) 973-761-9540
weitzrob@shu.edu

David Rosenthal
Seton Hall University
400 South Orange Avenue
South Orange, NJ, USA
(1) 973-761-9250
rosentdv@shu.edu

ABSTRACT

Intelligent tutoring systems have demonstrated their utility in a variety of domains. However, they are notoriously resource intensive to build. We report here on the development of a software tool that enables non-software developers to declaratively create intelligent tutors. This intelligent tutor generator creates applications with rich user interaction and powerful theory-based remediation capabilities. It utilizes the Constraint Based Tutoring paradigm and is generic enough to create tutors in several domains. It is easily extensible through plug-ins.

Categories and Subject Descriptors

K.3.1 [**Computer uses in education**]: Computer assisted instruction (CAI), Distance learning.

General Terms

Design, Human Factors.

Keywords

intelligent tutoring, constraint based tutors, model tracing tutors, instructional technology, distance learning

1. INTRODUCTION

Intelligent Tutoring Systems (ITS) represent a form of computer-based training in which the system uses a knowledge base to provide guidance to the student as the student interacts with the system. Such systems become increasingly relevant in the context of large amounts of training being delivered over the WWW. Building ITS has generally required significant software development expertise in addition to domain knowledge; authoring tools have been created to assist in this development [3]. This paper is an interim report on the architecture of an Intelligent Tutor Generator (ITG) based on the Constraint Based Tutoring paradigm [2]. We call it a tutor generator, as it enables the declarative creation of tutors in a variety of domains by non-software developers.

2. INTELLIGENT TUTORING

Intelligent tutoring is generally set in a problem-solving context. Tutoring is achieved through remediation provided while the student works on a problem. As the student interacts with the ITS by presenting either a partial or complete solution for evaluation, the ITS builds a student model (based on its own stored domain and problem-solving knowledge) and uses this to provide remediation. ITS are often seen as adjuncts to classroom teaching and hence usually do not incorporate features to teach concepts from scratch and in isolation. ITS can be ad-hoc, or based on formal theories of tutoring. Among ITS with theoretical underpinnings, Model Tracing Tutors [1] and Constraint Based Tutors are prominent.

3. VERSATUTOR – A TUTOR GENERATOR

Fig 1 shows the generic components of ITS. VersaTutor has generic versions of each of the components and can be used to create tutors in a broad variety of application domains. All the domain/problem specific information is supplied as data to VersaTutor. The system generates a problem-specific user interface based on the data. The constraint engine is generic and can handle constraints based on extensive string, numeric, logical and regular expression processing with numerous built-in functions that encompass the requirements of several domains. We created a powerful, flexible language with a formal grammar for expressing constraint conditions. Fig 2 shows the main screen of a VersaTutor session. It shows tutors from three domains: statistical hypothesis testing, elementary algebra and Java programming language syntax. VersaTutor is not currently web-based, but we have plans to web-enable it.

The student uses the “WorkArea” tab (Fig 3) to solve the problem. In our present implementation there are two kinds of interfaces supported for the work area, and the author of a tutor can choose the kind that is suitable for a problem. In the “Form” based user interface (shown in Fig 3), the student is presented with a list of available variables for the problem type in the drop down list on top. Selecting the variables appropriate for the specific problem from a master list of variables for the problem-type is one of the student’s tasks. The “Text” type of user interface, suitable when the student’s answer is a single text entity as opposed to a number of values typed into different fields, is shown in Fig 4. Fig 5 shows the situation after the student has submitted a partial solution (in the “Form” user interface) for evaluation and the tutor has provided feedback (Note that one of the variables is marked “Correct” in the “answer status” column, whereas the other variable is marked “Wrong”.) In case of errors, the tutor provides detailed feedback in the “Feedback” tab. It is quite common that the student’s submission violates several constraints.

4. CONCLUSIONS

This paper has described functionality and architecture of VersaTutor, a tutor generator that enables the creation of tutors with no programming and whose functionality can be enhanced through minimal programming. The base functionality is complete enough to provide rich interaction and remediation.

5. ACKNOWLEDGMENTS

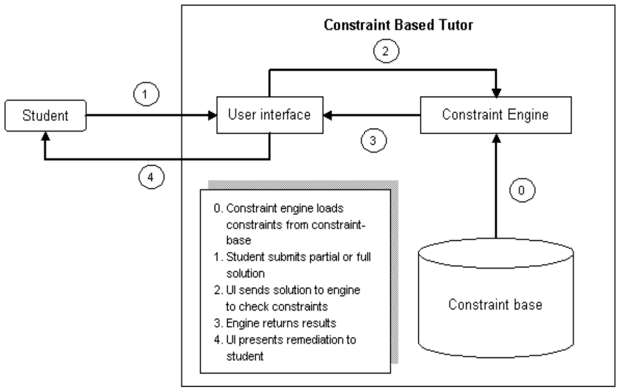
The authors wish to thank the Teaching Learning and Technology Center (TLTC) of Seton Hall University for having supported this research in the form of a CDI grant.

6. REFERENCES

[1] Anderson, J. R., Corbett, A. T., Koedinger, K., & Pelletier, R. (1995). Cognitive tutors: Lessons learned. *The Journal of Learning Sciences*, 4 (2), 167-207.

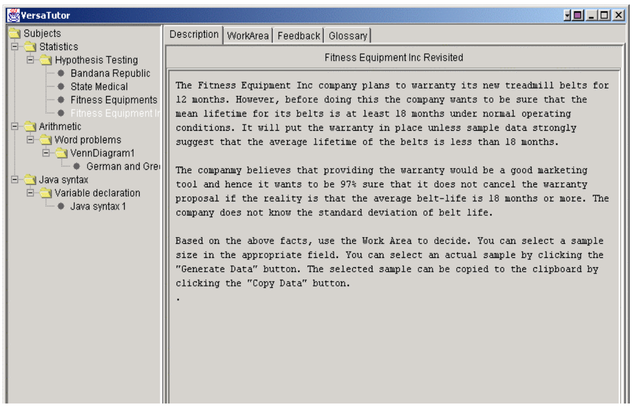
[2] Mitrovic, A., Mayo, M., Suraweera, P. and Martin, B., "Constraint-Based Tutors: A Success Story," in L. Monostori, JVAncza and M. Ali (Eds.), P*roceedings of the 14th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems (IEA/AIE – 2001),* pp. 931-940, Springer-Verlag, 2001.

[3] Murray, T., "Authoring Intelligent Tutoring Systems: An Analysis of the State of the Art," *International Journal of Artificial Intelligence in Education*, (10), 98-129, 1999.



0. Constraint engine loads constraints from constraint-base
1. Student submits partial or full solution
2. UI sends solution to engine to check constraints
3. Engine returns results
4. UI presents remediation to student

Figure 1



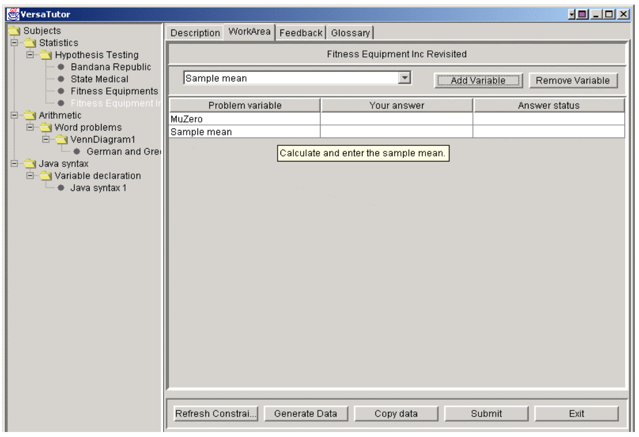
Fitness Equipment Inc Revisited

```
The Fitness Equipment Inc company plans to warranty its new treadmill belts for
12 months. However, before doing this the company wants to be sure that the
mean lifetime for its belts is at least 18 months under normal operating
conditions. It will put the warranty in place unless sample data strongly
suggest that the average lifetime of the belts is less than 18 months.

The companmy believes that providing the warranty would be a good marketing
tool and hence it wants to be 97% sure that it does not cancel the warranty
proposal if the reality is that the average belt-life is 18 months or more. The
company does not know the standard deviation of belt life.

Based on the above facts, use the Work Area to decide. You can select a sample
size in the appropriate field. You can select an actual sample by clicking the
"Generate Data" button. The selected sample can be copied to the clipboard by
clicking the "Copy Data" button.
.
```

Figure 2



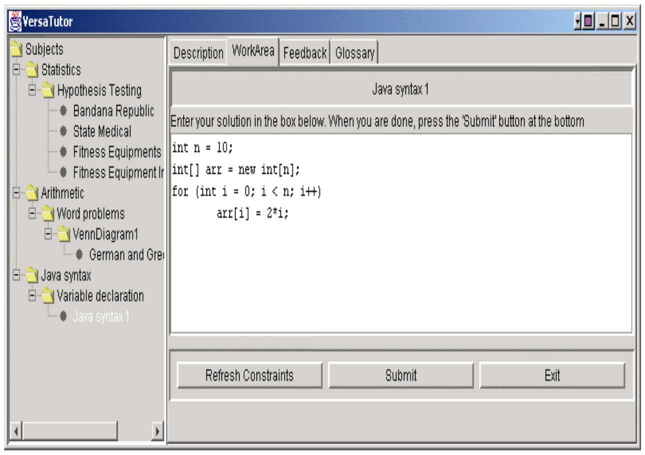
Fitness Equipment Inc Revisited

Sample mean

Problem variable	Your answer	Answer status
MuZero		
Sample mean		

Calculate and enter the sample mean.

Figure 3

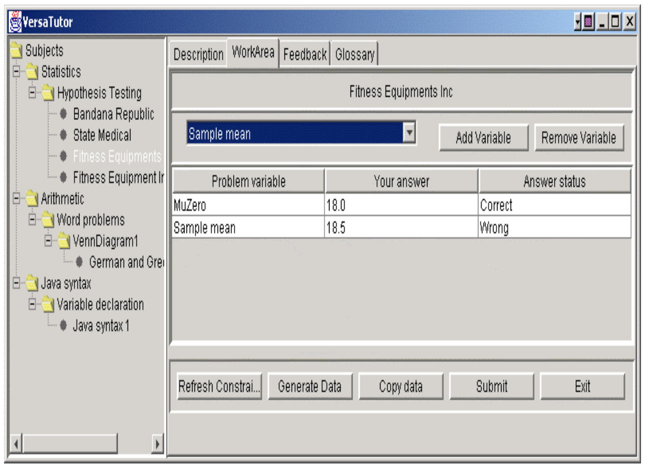


Java syntax 1

Enter your solution in the box below. When you are done, press the 'Submit' button at the bottom

```
int n = 10;
int[] arr = new int[n];
for (int i = 0; i < n; i++)
        arr[i] = 2*i;
```

Figure 4



Fitness Equipments Inc

Sample mean

Problem variable	Your answer	Answer status
MuZero	18.0	Correct
Sample mean	18.5	Wrong

Figure 5