*International Journal of Information Technology and Knowledge Management*
*July-December 2008, Volume 1, No. 2, pp. 351-363*

# TUTOR GENERATOR FOR INTELLIGENT TUTORING SYSTEM

**M. Siddappa, A. S. Manjunath * H. V. Ramakrishna**

**Abstract**

The emergence of Intelligent Tutoring System (ITS) has opened up new avenues for the use of computers in the field of education. ITSs are able to tackle difficult instructional problems and extend the usefulness of the computer as an instructional tool. Intelligent Tutoring Systems have proved useful in various domains, but are highly resource intensive to build, the "tutor generators" for assisting users in building ITSs conforming to two prominent intelligent tutoring paradigms: model-tracing and constraint-based. This paper presents design of a software tool called Intelligent Tutor Generators (ITG) for ITS. It is observed that it was easier to build a generic ITS generator based on the constraint-based paradigm, but the tutor generator based on the model-tracing paradigm is more feature rich. It can generate applications with rich user interaction and powerful theory-based remediation capabilities.

***Keywords:*** ITG, ITS, Paradigm

## 1. Introduction

Educational system in general and the learning process in particular are heading for a rapid change in the next millennium. Computers and computer networks are becoming major sources of information for the present and future generations.

Computers have made impressive strides in the field of education. As personal computers are becoming cost-effective with rapidly increasing capabilities, their application in education is becoming more and more prominent. Computers can provide individualized and interactive education for everyone. According to Skinner's theory of learning Skinner [1], education is nothing but the transmission of knowledge and can be done by breaking down any subject into a well-organized succession of modules and a question-answer process. A computer, being able not only to display matters, but also can interactively ask questions and grade answers. It can simulate and complement the teacher and allow each student to learn a subject and progress at ones own pace Hebenstreit, [2].

Intelligent tutoring is a knowledge-intensive activity. ITS have shown themselves to be effective in practice Anderson *et al*., [3] Mitrovic *et al*., [4]. They are, however, extremely resource intensive to build, requiring detailed knowledge about instructional

technology in general, the paradigm in use in particular, and a high-level of software development expertise. In an attempt to make development of ITS easier, tutor authoring tools or shells have been developed Murray [5].

This paper describes software tools - "Intelligent Tutor Generators" (ITGs) for assisting users in building intelligent tutoring systems.

- ITG focuses on
  - Constraint based paradigm.
  - Model tracing paradigm.

ITG for constraint-based tutors, entirely new tutors can be created quickly without any new software development since the constraints can be stated declaratively. ITG for model-tracing tutors requires the rules to be specified in JESS code Friedman-Hill *et al*., [6]. Both the ITGs are capable of generating problem-specific user interfaces based on declarative descriptions stored in a database.

Intelligent Tutor Generator's are used to build constraint-based and Model-Tracing Tutors for the statistical. hypothesis testing problem Kodaganallur et al., [7]. Currently MTTs are being built primarily by the Pittsburg Advanced Cognitive Tutor Center at Carnegie Mellon University and Constraint-Based Model Tutors (CBMT) at the Intelligent Computer Tutoring Group at the University of Canterbury, New Zealand. These groups have developed tools for generating tutors, Koedinger *et al*., [8] and Martin *et al*., [9], but we have found that these tools either use platforms that are not commonly available or are difficult to use without specific training. The constraint checker for our constraint based tutor is generic and can be used for several domains; it is also easily extensible.

The following sections cover theory-based approaches to intelligent tutoring, the two main paradigms of intelligent tutoring, the functionality of our tutor generators and their architectures.

## 2. INTELLIGENT TUTORING SYSTEMS

The necessary components of an ITS are the expert module, the student model module, the diagnostic module, the tutorial module and the user-interface module Siddappa et al.,[10]. The expert module in an ITS consists of domain knowledge that the system intends to teach the student. The expert module provides the necessary skill to the tutor to solve problems posed by the student and determines correct answers for the questions asked by students. The student model is that part of an ITS which represents the current knowledge state of the student. This information helps the tutor to adapt the instruction in accordance with competence, abilities and needs. The tutor can accordingly choose a suitable level and method of presentation of the subject based on the student's learning

abilities and other factors such as those represented in the student model. The main objective of diagnostic module is to maintain the student model and it does the evaluation of the student before, during and after the tutorial process. The information provided by the diagnostic module is to be used by the tutorial module to decide about what to teach and how to teach. The tutorial module contains instructional strategies like choosing an effective presentation method, determining what to present next and when to interrupt the instruction process. The instructional strategies are based on the information provided by the diagnostic module and the student module. The user-interface module provides communication between the student and the tutor which includes the actual presentation of text and graphics as well as acceptance of student input. A functional model of an ITS is given in figure 1

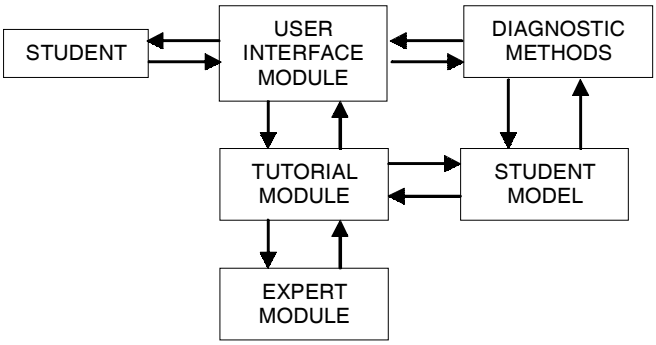


**Figure 1: Functional Model of ITS**

The constraint based paradigm is feasible only when the solution itself consists of sufficient information based on which the tutor can provide remediation. The model tracing paradigm is more effective in problem domains where there is a strong element of goal decomposition. Thus information content of the solution and the extent of goal decomposition are the two important dimensions which determine the suitable paradigm. Model tracing tutors have an added advantage in that they can provide guidance when a student is stuck with a blockade.

### 2.1. Constraint-Based Model Tutors (CBMT)

CBMT are rooted in theory; this theory stipulates that intelligent tutoring can be achieved by examining the problem state that a student arrived at and that the process the student used to reach that state is unimportant for remediation. The problem state is characterized by a-priori problem data and by variables for which the student supplies values while interacting with the tutor. At its core, this paradigm relies on the notion of constraint Mitrovic *et al*., [4], which can either be satisfied or violated. A problem state that violates one or more constraints is erroneous, and the violated constraints determine the remediation to be provided. A constraint is an ordered pair (Cr, Cs) of conditions with

Cr being the relevance condition and Cs being the satisfaction condition. Relevance and satisfaction conditions are Boolean expressions based on the problem state. A simple example is given below:

Cr: problem-type is "time and distance"

Cs: speed supplied by the student, equals distance divided by time.

An example from Ohlsson *et al*., [11] in the domain of algebra is:

*Cr* $(x + y)/d$ is given as the answer to $x/d_1 + y/d_2$

*Cs* $d = d_1 = d_2$

The relevance condition is used to determine whether a particular constraint is relevant to a problem state. The satisfaction condition of a constraint is evaluated if and only if the problem state ebmeets its relevance condition. If the satisfaction condition is met, then the constraint as a whole is said to be satisfied. A constraint whose satisfaction condition is not met is said to be violated, and suitable remediation may then be provided.

In the above example, the relevance condition is written in terms of the a-priori variable problem-type. The satisfaction condition is written in terms of both a-priori variables (distance and time) and a student-calculated value (speed). Evaluating a student's work involves checking the resultant problem state against every constraint, to determine all the constraints satisfied and violated. The corpus of violated constraints is used to determine the remediation. Fig. 2 shows the basic components of a constraint-based tutor and charts the interactions between these components in response to a student submission. Typical implementations would use highly specialized versions of each of the components suitable for the domain in question. Thus, for example, the SQL Tutor Mitrovic [12] provides a customized user interface, has a constraint engine with specific string processing capabilities needed for the domain and implements the constraint-base in LISP code. The need to build customized versions of each of the components for each application domain makes the creation of ITSs labor intensive.

### 2.2. Model-Tracing Tutors (MTT)

MTTs have been fielded in a variety of domains including college-level physics Gertner *et al*., [13], Shelby *et al*., [14], high school algebra Koedinger *et al*., [3], Heffernan *et al*., [15], Heffernan *et al*., [16], geometry Anderson *et al*., [17], Wertheimer *et al*., [18] and computer programming Corbett *et al*., [19], Corbett *et al*., [20], Corbett *et al*., [21].

This paradigm can be seen as taking a "process centric" view since it aims to understand the process that a student employs in arriving at a solution and to provide remediation based on this. An MTT is composed of expert rules, buggy rules, a model tracer and a user interface. Expert model rule is the step that a proficient individual

might take to solve the problem in question. These include rules for decomposing a problem into sub problems (or "planning" rules) and rules those address the solution of atomic sub problems ("operator" or "execution" rules). Planning rules embody procedural domain knowledge and operator rules embody declarative domain knowledge.

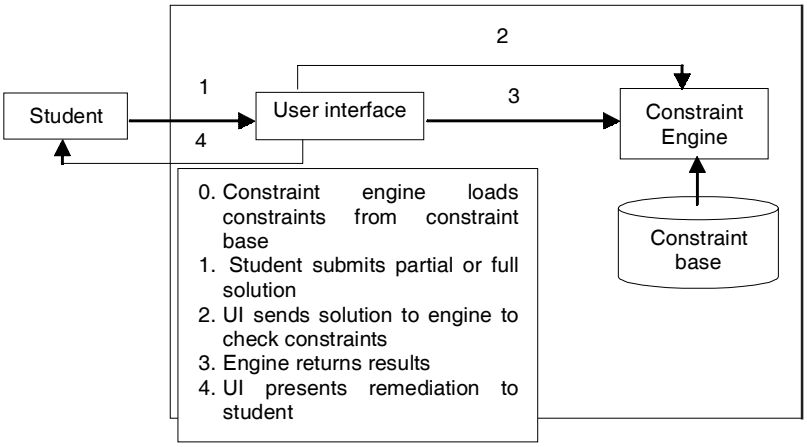


**Figure 2: Constraint Based Tutor Components and Interactions**

In an MTT, the components are similar to those in figure 2, except that the constraint base is replaced with a rule base, and the process of constraint checking will be replaced by a process of model- tracing.

The following is the example expert rule hypothesis testing domain. Hypothesis testing is a fundamental topic in inferential statistics Martin *et al*., [9]. A later section provides more details on this problem domain.

IF The goal is to solve the problem

THEN Set sub goals to determine:

The critical value, and

The sample statistic value.

IF The goal is to determine the critical value

THEN Set sub goals to determine:

The value of alpha, and

test statistic to use.

IF The appropriate test statistic to be used is the *z* statistic, and the population standard deviation is unknown.

THEN The value of the test statistic is

$$z = \frac{x - \mu}{s_x}, \text{ where } s_x = s_n$$

Here, the first two are planning rules and the third is an operator rule.

In MTT, domain knowledge is captured in the form of many such rules. The crux of an MTT is to "trace" the student's input, where tracing consists of finding a sequence of rule executions whose final result matches the student's input. If the tutor is able to do this, it is taken to mean that the tutor has understood the process by which the student arrived at an answer. In order to identify student errors an MTT has a set of "buggy" rules which reflect common student misperceptions. If the tutor's trace of a student solution contains the application of one or more of these buggy rules, then the tutor provides the remediation associated with the buggy rule(s).

A sample buggy rule in our tutor follows:

IF The population standard deviation is unknown, and the sample size < 30.

THEN The appropriate test statistic is the *z* statistic.

This rule models incorrect reasoning; for sample sizes less than 30 the appropriate test statistic is the *t* statistic. If the student's behavior matches this buggy rule, the system concludes that the student does not understand this piece of declarative knowledge, and provides the remediation associated with the rule. Since MTTs can provide well-targeted remediation only when one or more buggy rules are used in a successful trace, their tutoring capabilities depend on how well they capture the corpus of mistakes made by students.

While the ideal situation is that the tutor is able to trace all student inputs, this might not always be practical. Tutors should generally be able to trace all student inputs which are correct. For incorrect student input the tutor is unable to trace, the best remediation might be a general response that something is wrong.

In general, there could be several alternate strategies to solve a problem. A particular tutor might choose to allow only one strategy, or support multiple strategies. In the latter case, the tutor must have expert rules and buggy rules for each strategy, and the model-tracing process should be able to map the student's input to a particular strategy Schultze, K. G. [22].

## 3. USER INTERFACE

A different approach to building tutors looks at the possibility of creating generic versions of each of the requisite components and using these to help create tutors in a broad

variety of application domains. It is possible to do this fully for the constraint-based paradigm, but for the model-tracing paradigm it is unable to generalize the knowledge base that the tutor uses. Despite the limitations that generic ness introduces, we have seen that useful tutors from many domains can be quickly and easily generated from our generator.

When a generated tutor is started, a window appears showing the available problems. Our current implementations work at the level of individual problems; there is no sequencing of problems based on the student's performance. When a student selects a problem to work on, the Tutor displays a problem. The problem has several tabs and the student is free to move among the tabs as needed. To start working on the selected problem the student selects the "Work Area" tab. Solving a problem consists of selecting variables (from a master list) relevant to the problem solution and supplying values for those variables. Students type in values for the selected variables and submit the (possibly partial) solution for evaluation. The tutor evaluates the solution and provides solution in the answer status column. The tutor provides progressive hints, or the correct solution, on the student's request. If there are multiple errors, then the "Errors" list has multiple rows. When the student has supplied correct values for all the appropriate variables, the system treats the problem as having been completed. In the case of MTT, the "Guidance" tab and the "Guide" button are visible. When a student using the generated MTT is stuck and does not know how to proceed, then the student can press the "Guide" button and the tutor will provide guidance. Under the current state of the art in constraint-based tutors, such a facility is extremely difficult, if not impossible to provide in most domains. This is a natural feature of all model-tracing tutors since they always know where the student stands in the problem solving process. All the domain/problem specific information is supplied as data to our tools. The tools generate a problem-specific user interface based on declarative data. For someone wanting to use the generators without any additional coding, the user interface is currently restricted to allowing the student to select from a list of predefined variables and supplying values for these.

Anything else would require the tutor developer to write a plug-in. We planned to introduce a wide range of user-interaction primitives, using which tutor developers can declaratively create other user interaction paradigms.

## 4. CONSTRAINT-BASED TUTOR ARCHITECTURE

The other significant components are the constraint base and the constraint engine. All problem specific information is stored in a relational database. The generators support the notion of a problem type. Problem type represents a family of problems. For example, in the hypothesis testing domain, all single population tests for comparisons of means would constitute a single type. Thus once the constraints are specified for a problem type, any number of problem instances of that type can be created without needing to

specify the constraints all over again. A new problem can be added by populating database tables with all the problem parameters. The parameters problems are variable-value pairs. Some variables are a-priori variables which inform the tutor about some data that appears in the problem description. Other variables are the variables for which the student has to provide values. Tables 1 and 2 provide examples of a-priori and other variables.

**Table 1**
**A Priori variable Examples**

| A priori variable Name | Description |
|---|---|
| Problem Type | The kind of hypothesis testing problem(testing for mean less than a given value, greater than a given value or equal to a given value) |
| muZero | The hypothesized value of the population means |
| popStdKnown | Whether the population standard deviation is known |

**Table 2**
**Student Variable Example**

| *Variable Name* | *Description* |
|---|---|
| Null Hyp Sign | The comparison operator for the null hypothesis)<,> or =). |
| StudentmuZero | Students's value of mu zero. |
| Mean | The sample mean. |
| testStatistic | The test Statistic to be used |
| zValue | The value for the *z* statistic (this is applicable only if the proper statistic to be used in a problem instance is the *z* statistic). |

The constraint engine is generic and can handle constraints based on extensive string, numeric and logical processing, with numerous built-in functions that encompass the requirements of several domains. The constraint base is implemented with relational database technology and is independent of any programming language. A small constraint language enables tutor developers to specify constraints as Boolean expressions in a Java like syntax. The heart of the engine was developed using Java Compiler (Java CC) based on an EBNF specification of the language. Table 3 shows an example of a constraint. Note that the relevance and satisfaction conditions are shown as Boolean expressions. Associated with each constraint is the relevant remediation.

The generated tutors optionally allow for problem data to be generated individually for each student. That is, each student can work on the same problem, but with different randomly generated data. This is the case in the hypothesis testing problems shown on the screen. Data thus generated can be copied to the systems clipboard for transfer to any other application for further work.

**Table 3**
**A Constraint**

| *Problem type* | *Hypothesis Testing* |
|---|---|
| Constraint number | 30 |
| Description | If the problem type is "one population, testing for Mu<=muZero then the correct sign for the null hypothesis is "<=" |
| Relevance | Equals(problem Type, "IPM<=") &&exists("nullHypSign") |
| Satisfaction condition | Equals(("nullHypSign, "<=") |
| Error title | Incorrect null hypothesis |
| Hint 1 | Check your null hypothesis, You want to formulate your null hypothesis in such a way that you do not move away from the status quo unless there is strong evidence indicating otherwise. The null hypothesis is always that the observed mean favors retaining the status quo. |
| Hint 2 | Think above whether a larger value or a smaller value for the mean than muZero would favor moving away from the status quo, and then formulating the null hypothesis accordingly. In general, one would tend to stick with the status quo unless the data gathered convincingly indicates that there is a case to make changes, in this problem would a small sample mean or a large one convince you to maintain status quo? The null hypothesis is the case for maintaining the status quo. |
| Hint 3 | In the problem a small sample mean would convince you that it makes sense to stick with the status quo and not to take any action whereas a large mean would provide a rationale to do something. |
| Answer | The null hypothesis should be $\mu = \mu_0$ |
| Variable(s) | nullHyeSign |

The systematic interface between the user-interface and the constraint engine is very narrow. The user interface invokes a single operation on the constraint engine and passes the problem state to it. The result of this invocation contains all the information that the user interface needs to provide remediation to the student. This was explicitly done to facilitate the planned addition of a web interface.

Although the default functionality of the generator is already sufficient to build tutors in many domains, it is extensible through a plug-in architecture. New domain-specific functions can be added to the constraint language by writing standalone Java classes. The complete work area for a family of problem can be replaced by a user-supplied Java class that conforms to a specific interface. It is thus possible to create tutors with just the required functionality with a minimum of effort since most of the functionality will be reused.

## 5. MODEL-TRACING TUTOR ARCHITECTURE

We have already described the user interface portion of the MTT generator. The MTT uses the same approach as the constraint-based tutor generator for storing problems

and problem types. The major difference between the two is the knowledge base. In MTT the domain knowledge of the tutor is represented in the form of rules. The use of the Java Expert System Shell (JESS) Friedman-Hill, E [6] as our rule engine and coded the rules in the JESS language.

JESS is inherently a forward chaining rule engine. However, model-tracing requires a backward chaining rule engine. Fortunately JESS supports backward chaining through "backward chaining reactive" templates those allow one to build a goal structure and case the rule engine to try to achieve a goal by recursively trying to achieve its sub goals. An example snippet of JESS code that defines the goal breakdown is given in Figure 3. The first rule in that figure essentially says that in order to be able to make a decision, the critical value and the statistic value must both have been calculated. Firing this rule will cause JESS to try to satisfy the two sub goals. A JESS operator rule is shown in Figure 4.

```
(defrule decompose-ready-to-decide
        (Need-ready-to-decide)
        (Critical-value-computed)
        (statistic-value-computed)
        =>
        (assert(ready=to=decide)))
(defrule compute-right-side-cutoff-correct
        (need-critical-value-computed)
        (alpha-established)
        (statistic-established)
        (ht_prob (problemType "1PM<=") (testStatistic "z") alpha ?alpha) (zAlphaRight?
z&nil&(approx-eq?z (zlookup ?alpha))))
        =>
        (assert (cirical-value-computed))
        (addResult (fetch results) nil CORRECT (create$ "zAlphaRight") ) )
```

**Figure 3: Sample Goal Decomposition Rules in the MTT**

```
(defrule compute-right-side-cutoff-correct
        (need-critical-value-computed)
        (alpha-established)
        (statistic-established)
(ht_prob (problemType "1PM<=") (testStatistic "z") alpha ?alpha) (zAlphaRight
?z&nil&(approx-eq?z(zlookup ?alpha)))) => (assert(cirical-value-computed))
        (addResult(fetch results) nil CORRECT(create$ "zAlphaRight") ) )
```

**Figure 4: Sample Operator Rule Pertaining to Computing the Right Side Cutoff Value in the MTT**

In order to pass data back and forth between JESS and Java user interface uses the native data transfer mechanisms that JESS provides. Essentially data are passed back and forth as variable-value pairs. The MTT functions quite differently from the constraint-based tutor. Whereas in the constraint- based tutor, the student could submit values for any variables and get the partial solution evaluated by the tutor, the structure of model tracing is such that the tutor will proceed only along the goal structure. Suppose the student supplies values for a variable, which is part of a downstream goal without having satisfied upstream goals, the model-tracing process will find that earlier goals have not been satisfied. Hence some of the values supplied by the student will not be evaluated at all. The fact that these values were not evaluated itself tells the student that something is wrong.

The unique feature of MTT is their ability to provide guidance when the student does not know what to do next. In this case rather than submit a partial solution for evaluation, the student invokes the "Guide" button. At this stage the tutor knows exactly which goals have already been satisfied and which have not. Based on this the tutor can give precise guidance on what goals the student can try to meet next. This functionality is achieved by means of a set of "Guidance " rules stored in the rule base. An example guidance rule is given in Figure 5. Since a model tracing tutor can figure out the solution process the student knows which goals have been satisfied at any point in time. It provides remediation based on this knowledge. The example rule in Figure 5 is checking for the goal that is being satisfied (establishing the null hypothesis) and verifies that an upstream goal (computing mu-zero) has already been satisfied. It also verifies that the null hypothesis has not yet been established. Based on this information it constructs the guidance.

```
(defrule null-hypothesis-guidance
        (need- null-hypothesis –established)
        (mu-aero-computed)
        (ht_prob (nullHypSign nil)) => (store guidance
        (str-cat(fetch guidance) (build_guidance"decision" "hypotheses"
"mu_zero_established" "null_hypothesis") ) ) )
```

**Figure 5: Sample Guidance Rule in the MTT**

## 6. CONCLUSIONS

This paper describes tutor generators for building tutors based on the constraint- based and model-tracing intelligent tutoring paradigms. The generator for constraint-based tutors supports the creation of tutors from several domains without the need for any coding. The user interface, problem details and the constraints are all declaratively specified. The generator for model-tracing tutors also shares the same user interface capabilities, but its knowledge base is not generic and has to be coded in an expert system language as rules for each type of problem to be solved. On the other hand, the generated model-tracing

tutors have the ability to guide the student as to the possible next step when the student is lost. This capability is absent in the constraint-based tutors generated.

In this ITS it is possible to add features to the tutor generators. We are looking for ways to eliminate the need for tutor developers to have to write JESS code for creating MTT. We are also working on enhancing the user interface to provide many additional pre-built motifs that tutor builders can use directly to create more appealing interfaces.

## REFERENCES

[1] Skinner, B. F. . The Technology of Teaching. *Appleton Century Crofts,* New York, (1968).

[2] Hebenstreit, J. Computers in Education – the Next Step. Education and Computing, **1**(1), (1985), 37-43.

[3] Koedinger, K. R. and Anderson, J. R. Intelligent Tutoring Goes to School in the Big City. *International Journal of Artificial Intelligence in Education*, **8**, (1997), 30-43.

[4] Mitrovic, A. and Ohlsson, S. Evaluation of a Constraint-Based Tutor for a Database Language. *International Journal of Artificial Intelligence and Education*, **10**, (1999), 238–256.

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

[6] Friedman-Hill, E. Jess in Action: Rule-Based Systems in Java. Manning Publications, Greenwich, CT, (2003).

[7] Kodaganallur, V., Weitz, R., Rosenthal, D. V., A Comparison of Model-tracing and Proceedings of the 39th Hawaii International Conference on System Sciences – 2006 Constraint-based Intelligent Tutoring Paradigms, *International Journal of Artificial Intelligence in Education*, **15**, (2), (2005), 117–144.

[8] Koedinger, K. R., Aleven, V., and Heffernan, N.T. Toward a Rapid Development Environment for Cognitive Tutors. *12th Annual Conference on Behavior Representation in Modeling and Simulation.* Simulation Interoperability Standards Organization, (2003).

[9] Martin, B. and Mitrovic, A. WETAS: A Web-Based Authoring System for Constraint-Based ITS. In P. De Bra, P. Burilovsky and R. Conejo (Eds.), *Proceedings of the Second International Conference on Adaptive Hypermedia and Adaptive Web-Based Systems*, AH 2002, Malaga, Spain, 543-546, Berlin Heidelberg New York: Spring-Verlag, (2002).

[10] Siddappa. M, Dr. A. S. Manjunath, "Knowledge Representation using Multilevel Hierarchical Model in Intelligent Tutoring System", *Proceedings of the third IASTED International Conference, Advances in Computer Science & Technology*, April 2-4, 2007, Phuket, Thailand, ISBN CD:978-0-88986-656-0.

[11] Ohlsson, S. and Rees, E. The Function of Conceptual Understanding in the Learning of Arithmetic Procedures. *Cognition and Instruction*, **8**, (1991), 103–179.

[12] Mitrovic, A. An Intelligent SQL Tutor on the Web. *International Journal of Artificial Intelligence in Education* **13**, (2003), 171–195.

[13] Gertner, A. and VanLehn, K. Andes: A Coached Problem Solving Environment for Physics. In Gauthier, G., Frasson, C. and VanLehn, K. (Eds) Intelligent Tutoring Systems: 5th International Conference, ITS 2000, 131-142, Berlin Heidelberg New York: Springer-Verlag, (2000).

[14] Shelby R., Schulze K., Treacy D., Wintersgill M., VanLehn K., and Weinstein A. An Assessment of the Andes Tutor. *Proceedings of the Physics Education Research Conference*, 119-122, Rochester, NY, (2001, July).

[15] Heffernan, N. T., and Koedinger, K. R. (2002). An Intelligent Tutoring System Incorporating a Model of an Experienced Human Tutor. In S.A. Cerri, G. Gouardères, F. Paraguaçu (Eds.), Proceedings of the Sixth International Conference on Intelligent Tutoring Systems (ITS 2002), Biarritz, France and San Sebastian, Spain, June 2-7, 2002, 596-608, Berlin Heidelberg New York: Springer-Verlag.

[16] Heffernan, N.T. (2001). Intelligent Tutoring Systems Have Forgotten the Tutor: Adding a Cognitive Model of Human Tutors. *Doctoral Dissertation, School of Computer Science*, Carnegie Mellon University, Pittsburgh, PA 15213.

[17] Anderson, J.R., Boyle, C.F., and Yost, G. (1985). The Geometry Tutor. In A. K. Joshi (Ed.): *Proceedings of the 9th International Joint Conference on Artificial Intelligence*, 1-7, Los Angeles, CA.

[18] Wertheimer, R. (1990). The Geometry Proof Tutor: An "Intelligent" Computer-Based Tutor in the Classroom. Mathematics Teacher, 308-317 *Proceedings of the 39th Hawaii International Conference on System Sciences* 2006.

[19] Corbett, A. T. and Anderson, J.R. (1993). Student Modeling In An Intelligent Programming Tutor. In E. Lemut, B. du Boulay and G. Dettori (Eds.) Cognitive Models and Intelligent Environments for Learning Programming. Berlin Heidelberg New York: Springer-Verlag.

[20] Corbett, A. T., Anderson, J. R. and O'Brien, A. T. (1995). Student Modeling in the ACT Programming Tutor. In P. Nichols, S. Chipman and B. Brennan (eds.) Cognitively Diagnostic Assessment. (19-41). Hillsdale, NJ: Erlbaum.

[21] Corbett, A. T. and Bhatnagar, A. (1997). Student Modeling in the ACT Programming Tutor: Adjusting a Procedural Learning Model with Declarative Knowledge. *Proceedings of the Sixth International Conference on User Modeling*. Berlin Heidelberg New York: Springer-Verlag.

[22] Schultze, K. G., Shelby, R. N. Treacy, D. J., Wintersgill, M. C., Vanlehn, K., and Gertner, A. (2000). Andes: An Intelligent Tutor for Classical Physics. *Journal of Electronic Publishing*, 6, University of Michigan Press. (Retrieved June 14, 2005).

**M. Siddappa**
Research Scholar, Dept. of Computer Science & Engg.
Dr. MGR Educational and Research Institute, Chennai, Tamil Nadu, India
*E-mail: msiddu_ssit@rediffmail.com.*

**A. S. Manjunath**
Professor, Dept. of Computer Science & Engg.
Siddaganga Institute of Technology, Tumkur. Karnataka, India
*E-mail:asmanju@gmail.com*

**H. V. Ramakrishna**
Additional Rector
Dr. MGR Educational & Research Institute
Chennai, Tamilnadu, India