A Comparative Study of a Tool-Based Approach for Teaching Formal Specifications

Salamah Salamah
Embry Riddle Aeronautical University, ssalamah@erau.edu

Omar Ochoa, and Ann Q. Gates
University of Texas at El Paso, omar@miners.utep.edu, agates@utep.edu

Abstract - **The use of formal methods in software engineering has been shown to be an effective way to improving the dependability of developed systems. The difficulty of writing, reading, and understanding formal specifications, which is an essential component of formal verification, remains one of the main obstacles in the adoption of formal methods in industry. It is, thus, essential that undergraduate students in computing programs learn how to specify formal system properties. This paper outlines an instructive component that uses a tool to support the teaching of formal approaches and languages, and it presents educational outcomes from using such a component in undergraduate courses. The component uses a model checking-based tool LTLV to teach Linear Temporal Logic (LTL), a well-known specification language. In addition, this paper describes two experiments that compare the effectiveness of the new approach to that of the traditional lecture-based approach. The results of the experiments show promise that the tool-based approach can be effective in teaching formal specifications. The paper provides analysis of the results of the studies, as well as lessons learned from the experiments.**

Key Words – Model Checking, Formal Specifications, Linear Temporal Logic, Case Study

INTRODUCTION

The use of formal methods in software engineering has been shown to be an effective way to improve the dependability of developed systems. Formal verification techniques, such as model checking [3] and runtime monitoring [7], check the correctness of the system against properties specified in a formal language. Because formal specification languages are mathematically based, they offer the following advantages: they are unambiguous; they can capture precise pre- and post-conditions of methods that can be used to generate test cases [4]; and properties about the specifications themselves can be verified. The difficulty of writing, reading, and understanding formal specifications remains one of the main obstacles in the adoption of formal verification techniques. It is clear that there is a need to prepare a future workforce who can develop and verify systems, especially critical systems that require high assurance. The ability to apply the basics of formal specifications to specify software properties should be an outcome of computer science and software engineering programs.

This paper describes an instructive component that can be used in an undergraduate course to teach formal approaches and languages, and it presents the educational outcomes of such a component. The component uses a model checking-based tool called Linear Temporal Logic Validator (LTLV) to teach Linear Temporal Logic (LTL) [8], a well-known specification language for software and hardware systems. In addition, this paper describes an experiment that compares the level of understanding of LTL by three groups of students who had varying methods of instruction (lecture and tool-based) on the subject.

The experiment was conducted at two fundamentally different institutions, i.e., one was private and the other was public, in two fundamentally different courses and programs, i.e., one course was a capstone course in a computer science program and the other was an introductory course in software engineering in a software engineering program. All experiment subjects were novice in LTL.

The paper also reports on the use of the tool to improve the understanding of LTL of those students who have been previously introduced to LTL. The work compares the level of understanding of LTL by students in a senior level course in Software Quality Assurance before and after using the LTLV tool. Those students were previously introduced to LTL in a course on Formal Software Modeling.

The paper is organized as follows; the paper first provides the necessary background information on model checking and LTL. The LTLV tool is described in the following section. The paper then describes the instructive component and the new approach to teaching formal specifications followed by the description of the two experiments, and the results. The paper ends with a discussion of lessons learned and future work.

BACKGROUND

The following subsections provide the background information used in the rest of the paper. Specifically, we

summarize the concept of model checking and the LTL specification language.

Model Checking

Model checking is a formal technique for verifying finite or infinite-state concurrent systems by examining the consistency of the system against system specifications for all possible executions. The process of model checking consists of three tasks: modeling, specification, and verification.

The modeling phase consists of converting the design into a formalism accepted by the model checker. In some cases, modeling is simply compiling the source code representing the design. In most cases, however, the limits of time and memory mean that additional abstraction is required to come up with a model that ignores irrelevant details. In the SPIN model checker [5], the model is written in the Promela language [5], while in NuSMV [2] models are written in the SMV language [3].

Specification is the part of the model checking process where system properties are formally described. Properties are usually expressed in a temporal logic. The use of temporal logic allows for reasoning about time, which becomes important in the case of reactive systems. In model checking, specifications are used to verify that the system satisfies the behavior expressed by the property.

Once the system model and properties are specified, the model checker verifies the consistency of the model and specification. The model checker relies on building a finite model of the system and then traversing the system model to verify that the specified properties hold in every execution of the model [3]. If there is an inconsistency between the model and the property being verified, a counter example, in form of execution trace, is provided to assist in identifying the source of the error.

Linear Temporal Logic (LTL)

Linear Temporal Logic (LTL) [8] is a prominent formal specification language that is highly expressive and widely used in formal verification tools such as the model checkers SPIN and NuSMV. LTL is also used in the runtime verification of Java programs [6].

Formulas in LTL are constructed from elementary propositions and the usual Boolean operators for not, and, or, imply (!, &, |, → respectively). In addition, LTL provides the temporal operators next (X), eventually (F), always (G), until, (U), weak until (W), and release (R). These formulas assume discrete time, i.e., states s = 0, 1, 2,...The meanings of the temporal operators are straightforward[1]:

- The formula Xp holds at state s if p holds at the next state $s + 1$
- The formula p U q is true at state s, if there is a state $s' \geq s$ at which q is true and, if s' is such a state, then p is true at all states s_i for which $s \leq s_i < s'$
- The formula F p is true at state s if p is true at some state $s' \geq s$
- The formula G p holds at state s if p is true at all states $s' \geq s$

A problem with LTL is that the resulting LTL expressions can become difficult to write and understand. For example, it is hard to deduce that the LTL formula "G (a → F (p & F ((!p) & !a)))" represents the English requirement "If a train is approaching (a), then it will be passing (p), and later it will be done passing with no train approaching".

LTLV TOOL[2]

The LTL Validator (LTLV) tool was developed at the Department of Computer Science at the University of Texas at El Paso. The purpose of the tool is to allow for the validation of LTL formulas by users through examining the behaviors accepted and rejected by the LTL specification. The tool interacts with a model checker to examine the different behaviors of an LTL formula.

As opposed to using a model checker to test the correctness of the model, the technique [9, 10] used by LTLV uses a simple model written in SMV [3] to test whether an LTL specification holds for a given trace of computation. A *trace of computation* is a sequence of states that depicts the propositions that hold in each state. For example, in the trace " - - - P - - R - (Q S) - -", proposition P holds in state 4, R holds in state 7, and both Q and S hold in state 9.

The user models a trace of computation by assigning truth values to the propositions of the LTL formula for a particular state. For example, a user may examine one or more combinations of the following: a proposition holds in the first state, a proposition holds in the last state, a proposition holds in multiple states, a proposition holds in one state and not the next …etc. This assignment of values is referred to as a test. The user runs the LTLV tool code, the test case, and the LTL specification. Each run assists the user in understanding a formula by checking expected results against actual results. The simplicity of the model makes inspection of the result feasible.

The (LTLV) tool, as pictured in Figure 1, consists of: two input boxes labeled *LTL Formula* and *Traces*; a radio button labeled *Expected Result*; a button labeled *Check*; and an output box that displays the results of the validation.

Users of the tool enter LTL formulas into the input box labeled *LTL Formula*, following the operator syntax as described in the *LTLV HELP* window, as shown in Figure 2. In addition, users enter a trace of computation against which the formula will be tested. The *LTLV HELP* window

[1] We only provide the meaning for those operators used in model checking.

[2] The LTLV tool can be requested from any of the authors.

describes examples of formula input, traces, and counter traces. The user selects from the *Expected Result* radio buttons whether the result is expected to evaluate to *true* or *false* for the formula applied to the trace. Upon the user pressing the *Check* button, the tool displays the results of the validation.

The tool interfaces with the NuSMV model checker. The LTL formula and the provided trace are passed to NuSMV, which returns a *true* or *false* value representing if that trace satisfies the given LTL formula. The resulting value is compared with the *Expected Result*; if they match, the tool reports that the user has passed the test otherwise, the tool reports that the test has failed.

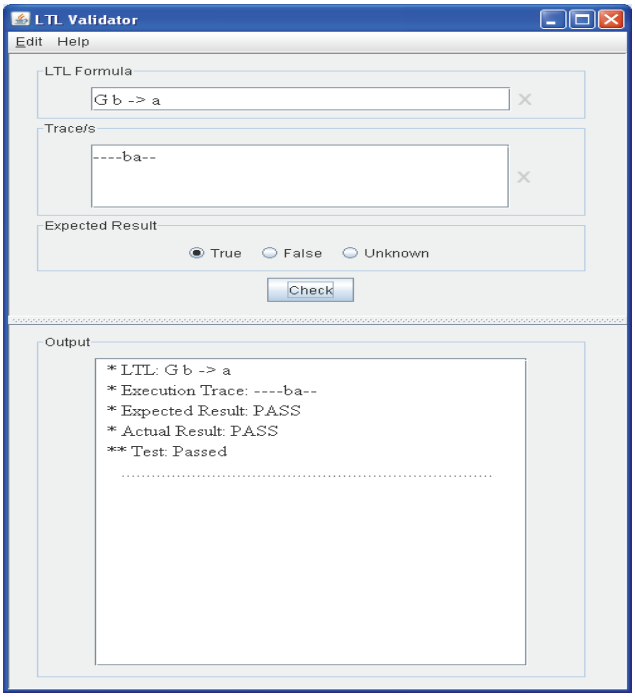

FIGURE 1
MAIN WINDOW OF THE LTLV TOOL

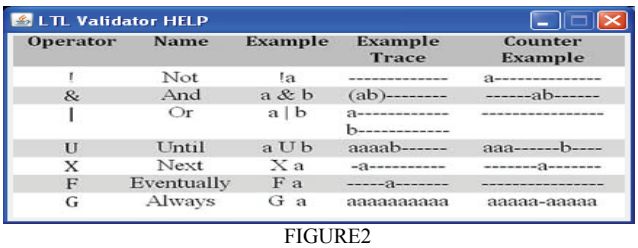

FIGURE2
HELP WINDOW OF THE LTLV TOOL

AN EDUCATIONAL COMPONENT FOR TEACHING FORMAL SPECIFICATIONS

The goal of the lessons described in this section are to teach students how to: 1) write, read, and understand formal specifications using LTL, and 2) use tools (model checking-based tools) in support of specifying software properties and analyzing the generated specifications using these tools. This section describes a technique and activities that support the attainment of these goals. The prerequisite for the educational component is a course in discrete mathematics in which the students specify properties using propositional logic. The component described here is approximately six hours in length with tutorials on using the tools.

The educational outcomes, given below, are separated using Bloom's taxonomy [1], where level 1 outcomes represent knowledge and comprehension outcomes (those in which the student has been exposed to the terms and concepts at a basic level and can supply basic definitions); level 2 outcomes represent application and analysis (those in which the student can apply the material in familiar situations); and level 3 represent synthesis and evaluation (those in which the student can apply the material in new situations). The educational outcomes are:

1-1. Students will be able to describe the behavior of simple LTL formulas.
2-1. Students will be able to use a model checking-based tool to determine if properties hold in a model.
2-2. Students will be able to use a model checking-based tool to improve their understanding of LTL.
3-1. Students will be able to specify a property in LTL and be able to define test cases to test the property.

In this paper, we only show the lessons and exercises for outcomes 2-2 and 3-1. A complete description of lessons and exercises for the other outcomes can be found in Salamah et.al, [10].

Exercises Using LTLV to Improve Understanding of Formal Specifications

The following exercises illustrate the use of the LTLV tool in teaching formal specifications.

Exercise 1: The focus of Exercise 1 is to clarify the subtleties of the temporal operators of LTL (Outcome 2-2). For example, the LTL formula "*F P*" asserts that *P* holds at some future state; however, in LTL, the current state is part of the future and, hence, the situation where *P* holds in the current state is accepted by this specification. Instructions for Exercise 1 follow:

The Response property asserts that P and S are related in a cause-effect manner (P causes S). The LTL formula for the response property is $(P \rightarrow F\ S)$. *Use LTLV to run the traces of computation given below against the specified LTL formula to answer the following questions: (1) Does the Response property guarantee the occurrence of effect (S) in all cases? Explain your answer. (2) In the response property, can the cause (P) and effect (S) hold at the same state? Explain your answer.*

(1) - - - - - - - - -
(2) - - - - - - - - P
(3) - - - - - - - - S
(4) - - - -(PS) - -

Exercise 2: Exercise 2 focuses on teaching students the concepts of traces of computations, and how they can be used to visualize the appropriate behaviors accepted by an LTL formula. In addition, the subtle properties of LTL are explained. This exercise also targets Outcome 2-2. Instructions for Exercise 2 follow:

Consider the following property: "When a connection is made to the SMTP server (P), all queued messages in the OutBox mail will be transferred to the server (R)", and it representative LTL formula: (G !R) | (!R U (P & !R))

For each trace of computation given below, state the expected result of running LTLV. Check your results by running LTLV as shown in class to validate your predicted results. Finally, state whether the LTL specification matches the natural language property. Explain your answer.

(1) - - - - - - - - - - - -
(2) - - - - - (P R) - - -
(3) - - - - -P R - - - - -
(4) - - - - - R P - - -
(5) - - - - - - - - - - R
(6) R - - - - - - - - - -
(7) - - R - - P - R - -

Exercise 3: After completing Exercises 1 and 2 as well as other assignments determined by the instructor, the students should be able to define more sophisticated LTL specifications. In addition, they should be able to define test cases in the form of traces of computations to validate their generated LTL formulas. Exercise 3 checks their ability to satisfy Outcome 3.1. Instructions for Exercise 3 follow:

For each of the properties given below, specify the corresponding LTL formula. In addition, define a minimal set of traces of computations to validate your generated LTL formula. Use LTLV to test each trace of computation against your generated formula. The Properties are:

(1) *When a connection is made to the SMTP server, all queued messages in the OutBox mail will be transferred to the server.*
(2) *When the name of a mailbox is double-clicked, the mailbox will be opened.*
(3) *The OK button on the login window is enabled as soon as the login window is first displayed to the user.*

EXPERIMENTS TO VALIDATE EFFECTIVENESS OF NEW TEACHING APPROACH

This section describe the setup of two experiments to validate the effectiveness of using LTLV and the new approach described in the previous section in teaching LTL.

The first experiment was conducted at the University of Texas at El Paso (UTEP), and Embry Riddle Aeronautical University (ERAU) in Daytona Beach Florida. Experiment subjects were undergraduate students in Computer Science (UTEP) and Computer Science, Computer Engineering, Software Engineering, Aeronautical Science, and Aerospace Engineering (ERAU). Students from UTEP were all seniors, while those from ERAU ranged from juniors to seniors. None of the participants had previous knowledge of LTL. It is assumed that all participants had the same level of knowledge in logic and formal languages through the typical course in discrete mathematics taught in both universities and as prerequisites for both courses (at UTEP and ERAU).

The second experiment was conducted using students in a senior level Software Quality Assurance course at ERAU. The students had previous knowledge of LTL through a course in Formal Software Modeling

It is important to note that the experiments do not compare the whole approach provided in Salamah et.al, [10]. They were only intended to provide some evaluation of the effectiveness of the tool-based teaching element of that approach.

Experiment I Procedure

The following procedure was followed for Experiment I in both groups at UTEP and ERAU.

The initial step in this study was to give an introductory lecture on LTL to all participants. The lecture covered the motivation for temporal logic, different types of temporal logics, the formal definition of LTL operators, and examples of LTL specifications using these operators. The introductory LTL lecture was delivered by the same instructor (this Paper's first author) at both UTEP and ERAU.

The first lecture was followed by an initial test on LTL. The test consisted of two types of questions. On the first question the students were given an LTL formula and a set of traces of computations and were asked to describe whether each trace represents an acceptable behavior by the LTL specification (similar to Exercise 2 above). On the second question, the students were given a natural language description of a property and a set of LTL formulas and were asked to select the LTL formula that best describes the natural language property. Students were asked to complete their answers on two answer sheets. One sheet was returned to the experiment organizers, while the other was kept with the student for further use.

Once the students completed the initial test they were divided into three groups. Students in the first group (henceforth called *home group*) were asked to leave and needed no further participation other than taking a final exam on LTL at a later time. Students in this group were not given access to the LTLV tool.

Students in the second group (henceforth called *lecture group*) were gathered again with the instructor and went back over their answers to the initial exam. Students in this group were not given access to the LTLV tool.

Students in the third group (henceforth called *tool group*) were given a 15 minute tutorial of the LTLV tool. Students in this group were then asked to use the tool to examine their answers for the initial exam questions. The

goal was for this group to use the tool to learn the subtleties of LTL.

Students in each of the three groups were asked to take a final test on LTL three days after the initial test. The format and number of questions on the final LTL test were the same as those on the initial LTL test.

Experiment I Results

Table I shows the minimum, maximum, and average grades for the initial exam, for each group type at each institution[3]. Table II shows the minimum, maximum, and average grades for the final exam, for each group type at each institution.

TABLE I
EXPERIMENT I RESULTS FOR INITIAL EXAM

Institution	Group Type	Min	Max	Mean
A	Home	38%	75%	59%
A	Lecture	38%	100%	70%
A	Tool	44%	94%	64%
B	Home	44%	88%	65%
B	Lecture	38%	56%	49%
B	Tool	38%	88%	61%

TABLE II
EXPERIMENT I RESULTS FOR FINAL EXAM

Institution	Group Type	Min	Max	Mean
A	Home	18%	94%	55%
A	Lecture	35%	88%	63%
A	Tool	35%	94%	60%
B	Home	35%	76%	51%
B	Lecture	29%	94%	64%
B	Tool	29%	94%	61%

Analysis of Experiment I Results

The results of the examination show that students at institution A performed worse when comparing the initial exam averages to the final exam averages, in contrast, students at institution B performed better on their exam averages. This difference can be attributed to the fundamental differences between the group of students at each institution, i.e., the courses and programs that the students are part of. Additionally, students at institution A were told that the grades on these exams would not affect their course grade, therefore students at institution A studied minimally, if at all; while students at institution B were told that they could receive extra credit based on their performance on the final exam.

It is interesting to note that the final exam grades for both institutions are consistent between same groups at different institutions, i.e., the highest grades come from the *lecture group*, then the *tool group*, and finally the *home group*.

Group Comparison for Experiment I

At institution A, the initial exam grade for the *home group* is 59% which dropped to 55% on the final exam grade. In comparison, the exam grade average for the *lecture group* is 70% and dropped to 63% for the final exam. For institution B, the initial exam grade for the *home group* is 65% which dropped to 51% on the final grade. In comparison, the exam grade average for the *lecture group* is 49% and increased to 64% for the final exam. Of particular interest is that the groups with the highest initial averages, institution A-*lecture group* and institution B-*home group*, had the biggest drops on the final exam average grade. This can be attributed to the fact that the students felt confident of their high scores and did not prepare adequately for the next exam.

The *tool group* at institution A demonstrated to be the best group at that institution at retaining knowledge, as the drop between exams was the lowest of the three, at only 3%. At institution B, for the *tool group* the grades between both exams were the same, at 61%, outperforming the *home group* and being close to the 64% final exam average of the *lecture group*. Much like the lecture approach, the tool approach is better than the approach in the *home group*, but the tool approach exhibits a greater degree of knowledge retention.

Experiment II Procedure and Results

Experiment II was conducted using the students in a Software Quality Assurance Course at ERAU. The class had 12 students. All students had previous knowledge of model checking and LTL through a course in Formal Software Modeling. All 12 students were enrolled in both classes at the same semester. The topics on temporal logic were covered in the Formal Modeling class about two weeks prior to this experiment

Students in this Experiment were treated in the same way as those in the *Tool group* in Experiment I. Students were given the same initial lecture and initial test on LTL as in the previous Experiment. Once again, the introductory LTL lecture was delivered by this paper's first author. The students were then given a 15 minute tutorial on using LTLV. Finally, the students were given the same final test on LTL as in Experiment I. The final test was given three days after the initial lecture and test. Both initial and final tests were part of the final students' grade for the semester. Students were given access to the LTLV tool and were encouraged to use it in preparation for the final test on LTL.

Table III shows the minimum, maximum, and average grades for the students' initial (Exam I) and final (Exam II) exams.

TABLE III
EXPERIMENT II RESULTS

Exam I MIN	Exam II MIN	Exam I MAX	Exam II MAX	Exam I MEAN	Exam II MEAN
33%	42%	88%	100%	60%	72%

The results in Table III show that the students' performance improved after using the LTLV tool. Maybe as important to note is the fact that when students completed an end of semester course evaluation, 67% of the students (8 out of 12) indicated that they felt that the use of LTLV improved their understanding of LTL and vowed to use it as

[3] Institution A is UTEP and B is ERAU.

part of their preparation for their final examination in the Formal Modeling Course.

SUMMARY AND LESSONS LEARNED

One of the issues in employing formal verification techniques is the difficulty in writing, reading, and understanding formal specifications. Because of the increased use of formal verification techniques in software development there is a need for software engineers to understand the subtleties of formal specifications and to gain experience in formal methods, especially for those working with high-assurance systems.

This paper describes a technique that assists in the understanding of LTL through the use of a model checking process. The technique uses the LTLV tool to elucidate the behavior of an LTL formula and promote understanding by analyzing the property (represented by the formula), test case (that represents a pre-defined trace of computation), and anticipated and actual result. The paper presents educational outcomes and exercises to support using the technique. Although the technique is being used to teach LTL, it can easily be adopted to other formal languages such as the Computational Tree Logic (CTL) [3].

The paper also presented two experiments to validate the effectiveness of the new approach. The first experiment compared the effectiveness of the teaching approach using the LTLV tool to that of the traditional lecture-based approach. The results of the experiment showed that the tool-based approach did as well as the lecture approach.

Considering that the subject being taught (LTL) and the type of teaching were completely new to the students, we anticipate that more teaching using this tool-based approach would yield better results in the future. This is evident by the fact that students who were already introduced to LTL performed better when using the tool as shown by the second experiment.

Lessons Learned and Future Work

During our analysis of the results of the students' scores in Experiment I, we noticed that ERAU students' scores on the final test did not differ a great deal from their scores on the initial test. This was not the case for the students at UTEP. We believe that this is caused by the fact that students at ERAU were told that their scores on the initial and final LTL tests will be counted toward their final exam grade for the class. We believe that this provided an incentive for students at ERAU to do better on their tests.

Future work includes conducting a similar experiment taking the following into consideration; 1) students motivation should be the same across all groups, 2) students should be divided into groups of similar academic achievements (based on GPA or current class average), and 3) We might want to consider examining the type of problems that are best tackled by traditional lectures and those that benefit most from our tool-based approach. Additionally, we plan on improving the LTLV tool to provide more description or suggestions as to why a test case has failed.

ACKNOWLEDGMENT

This work is partially funded by the National Science Foundation grants: Cyber-Share Center of Excellence: A Center for Sharing Cyber-resources to Advance Science and Education (HRD-0734825), and Computing Alliance of Hispanic-Serving Institutions CAHSI (CNS-0540592).

The authors would like to thank students from UTEP and ERAU who participated in both studies. The authors would also like to thank the reviewers of the initial submission for their valuable comments

REFERENCES

[1] Bloom, B.S., "Taxonomy of Educational Objectives: The Classification of Educational Goals," Susan Fauer Company, Inc., 1956, pp. 201-207.

[2] Cimatti, A., Clarke, E., Giunchiglia, F., and Roveri, M., "NuSMV: a new Symbolic Model Verifier", *11th Int'l Conference on Computer Aided Verification, CAV*, Trento, Italy, July 1999, pp. 495-499.

[3] Clarke, E., Grumberg, O., and D. Peled. *Model Checking*. MIT Publishers, 1999

[4] Cheon Y., Leavens G. T., "A Simple and Practical Approach to Unit Testing: The JML and JUnit Way", *European Conference on Object-Oriented Programming, ECOOP*, Malaga, Spain, June 2002.

[5] Holzmann, G.J., *The SPIN Model Checker,* Addison-Wesley, 2004.

[6] Havelund, K., and Pressburger, T., "Model Checking Java Programs using Java PathFinder", *Int'l J. on Software Tools for Technology Transfer*, 2(4), 2000.

[7] Gates, A., Roach, S., et al., "DynaMICs: Comprehensive Support for Run-Time Monitoring," *Runtime Verification Workshop*, Paris, France, July 2001, pp. 61-77.

[8] Manna, Z. and Pnueli, A., "Completing the Temporal Picture," *Theoretical Computer Science,* 83(1), 1991, 97-130.

[9] Salamah, S., Gates, A., Roach, S., and Mondragon, O., "Verifying Pattern Generated LTL Formulas: A Case Study." Lecture Notes in Computer Science (LNCS) Volume 3639/2005, Springer-Verlag, pp. 200-220, 2005

[10] Salamah, S., and Gates, A.,"A Technique for Using Model Checkers to Teach Formal Specifications" Proceedings of the 21st IEEE-CS International Conference on Software Engineering Education and Training (CSEET), Charleston, SC, April 2008, 181-188.

AUTHOR INFORMATION

Salamah Salamah, Assistant Professor, Department of Electrical Computer Software and Systems Engineering, Embry-Riddle Aeronautical University.

Omar Ochoa, Graduate Research Associate, Department of Computer Science, University of Texas at El Paso.

Ann Q. Gates, Professor of Computer Science and Associate Vice-President, Office of Research and Sponsored Projects, University of Texas at El Paso.