UNIVERSITY OF WESTMINSTER

WestminsterResearch
http://www.wmin.ac.uk/westminsterresearch

An e-learning tool for understanding schedule properties.

Steve Barker
Department of Computer Science, King's College, London

Paul Douglas
Cavendish School of Computer Science, University of Westminster

Copyright © [2003] IEEE. Reprinted from ITCC 2003: International Conference on Information Technology: Computers and Communications. IEEE Press, pp. 53-59.

This material is posted here with permission of the IEEE. Such permission of the IEEE does not in any way imply IEEE endorsement of any of the University of Westminster's products or services. Internal or personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution must be obtained from the IEEE by writing to pubs-permissions@ieee.org. By choosing to view this document, you agree to all provisions of the copyright laws protecting it.

The WestminsterResearch online digital archive at the University of Westminster aims to make the research output of the University available to a wider audience. Copyright and Moral Rights remain with the authors and/or copyright owners. Users are permitted to download and/or print one copy for non-commercial private study or research. Further distribution and any use of material from within this archive for profit-making enterprises or for commercial gain is strictly forbidden.

Whilst further distribution of specific materials from within this archive is forbidden, you may freely distribute the URL of WestminsterResearch. (http://www.wmin.ac.uk/westminsterresearch).

In case of abuse or copyright appearing without permission e-mail wattsn@wmin.ac.uk.

An E-Learning Tool for Understanding Schedule Properties

Steve Barker,
Department of Computer Science,
King's College, London, UK.
email: steve@dcs.kcl.ac.uk

Paul Douglas,
Cavendish School of Computer Science,
Westminster University, London, UK.
email: P.Douglas@wmin.ac.uk

Abstract

In this paper, we describe an e-learning tool that we have developed to assist University students studying various modules on database systems. We use the acronym DTST (viz. a learning tool for Database Transaction Schedule Testing) to refer to our learning tool. DTST enables students to actively construct their own learning environment, it can respond in an individualistic way to student input, and it has a built-in web interface that makes it widely accessible. Field tests conducted on DTST suggest that it provides students with a different and valuable type of learning experience that traditional methods do not provide.

1. Introduction

We describe an item of educational software that we have developed and used to help University-level students to learn certain key notions in database transaction processing. We call this piece of software DTST (*viz.* a learning tool for Database Transaction Schedule Testing).

DTST is an item of educational software that is intended to "intelligently" assist computer science students in developing their understanding of *CRAS property* satisfaction [2]. The term "intelligently" is interpreted by us as the capability of responding to a student's self-selected input by detecting, diagnosing and explaining his/her errors or confirming that his/her understanding is correct.

DTST is a learning aid that is able to respond to questions about CRAS property satisfaction in the same way that an "expert tutor" might and provides students with a tool for constructing their own learning experience, an attractive feature that textbooks do not provide. More specifically, DTST encourages students to learn about the CRAS properties by making and testing hypotheses. This approach appears to us to be the approach students naturally adopt to learn about the CRAS properties. The traditional, text-based method that we have previously used to teach the CRAS properties does not adequately support learning by hypothesis formulation and testing.

Although DTST has thus far been used to help students to learn about CRAS property satisfaction, DTST may be modified to provide support for students learning any part of the Computer Science curriculum that involves reasoning about the consequences of the occurrences of events.

The rest of the paper is organized thus. Section II introduces the CRAS properties. In Section III, the development of DTST is discussed and some key features of the software are outlined. Section IV gives details of the web interface. In Section V, some results, produced from the formative and summative evaluations of DTST, are described and discussed. In Section VI, conclusions are drawn, and suggestions are made for further work.

Due to space limitations, in this paper we only give an overview of our work and we sketch the main results. A fuller set of results and a discussion of them will be presented in a forthcoming publication.

2. The CRAS Properties

The CRAS properties are conditions imposed on a *schedule*, a sequence of interleaved read and write operations performed on objects in a database. Schedule operations are performed by a *database management system (DBMS)* as part of a *transaction* [2].

Unfortunately, certain interleavings of the operations from different transactions in a schedule can cause anomalous behaviours (that compromise database integrity) and can raise a number of practical difficulties (e.g., the *lost update problem* [3]). The CRAS properties [2] solve this type of problem by imposing constraints on the order in which operations are performed in a schedule; satisfaction of these constraints guarantees that schedules are correct. Moreover, the DBMS can be configured to optimize the performance of transaction processing.

The CRAS properties are: conflict serializability, recoverability, avoids cascading aborts and strictness [2].

Proceedings of the International Conference on Information Technology: Computers and Communications (ITCC'03)
0-7695-1916-4/03 $17.00 © 2003 IEEE

IEEE COMPUTER SOCIETY

3. DTST: An Overview

DTST is a piece of software that enables students to test any syntactically correct schedule they choose as input to the system. Students also have complete freedom to choose to investigate the satisfaction of any of the CRAS properties by these schedules.

The software that implements DTST is written in PROLOG [4]. PROLOG has been widely used for implementing items of educational software (see, for example, [13]) and is appropriate for developing applications, like DTST, that require that a degree of "intelligence" be captured. The fact that the rules that define the CRAS properties can be directly translated into PROLOG was another reason for us choosing PROLOG to implement DTST.

3.1. Our Development Methodology

Our approach to developing DTST initially involved us adopting a *phenomenographic* method [11] for information gathering on students' understanding of concepts in transaction processing. By conducting 'dialogue' sessions with students we identified the strategies students used to understand the CRAS properties. From our review of the notes taken at the dialogue sessions, we were able to develop a prototype system for supporting students in learning about CRAS property satisfaction.

As our DTST tool evolved, we made increasing use of Gagne's event-based model of instruction [9] to decide what material a user of DTST should be offered and the order in which information ought to be presented to a learner. Following [9], when students use DTST they are reminded what the learning task to be performed is, and what it is they are supposed to be able to do once the learning task has been completed. Prominence is given to the distinctive features that need to be learned, different levels of learning guidance are supported for different types of learners, informative feedback is given, and learning takes place in a student-centred, interactive way, but with support available to students as and when they need it.

3.2. A Learning Session

When engaging with DTST, a user submits a schedule σ to the system and selects a CRAS property to evaluate with respect to σ. The schedule is displayed to the user who may then hypothesise about the satisfaction or otherwise of any of the CRAS properties by σ and may test these hypotheses by using DTST. Moreover, the user may request explanations of the answers that DTST generates. Each operation in σ is represented by a 4-ary tuple, (o,t_j,i,t_s). Here, o denotes an operation (e.g., read or write), t_j denotes a transaction performing o, i denotes the data item acted upon by t_j, and t_s is the time at which o is performed. In the case where o is a commit or an abort, the data item is *null* as these operations are not performed on a data item. Hence, the output (in *italics*) produced in a DTST session might be:

Your chosen schedule was:

write, tr1, dataX, 1
write, tr1, dataY, 2
read, tr2, dataU, 3
write, tr1, dataZ, 4
write, tr2, dataZ, 5
commit, tr1, null, 6
write, tr2, dataX, 7
read, tr2, dataY, 8
write, tr2, dataY, 9
commit, tr2, null, 10

An example query (chosen from a menu) on this schedule is:

Correct for ACA? (i.e., the user asks "is this schedule an ACA[1] schedule?").

In this case, the output DTST produces is "*yes*".

Thereafter, a user can ask DTST for an explanation of this result by selecting the *explain* option from a menu.

All of the CRAS properties and schedules are evaluated in a similar way and several levels of explanation are provided by DTST. DTST also includes a number of on-line tests for students to check their understanding of the CRAS properties. DTST is able to automatically analyse student responses to the on-line test questions and can diagnose misunderstandings and report them to the student.

4. The Internet Interface

We have used CGI [12] as our interface mechanism. The principal reason for this is that it was easy to install on the University system, with good support readily available. In addition, CGI's simplicity enables the applications to ignore the problem of getting their output to a web server, so no significant modifications of the fundamental application were required.

We have used Eugene Kim's CGIHTML package [6], rather than something like Apache's API, for its portability. All development work was done on Solaris using Sun's Forte compiler and an Apache [1] server.

Our Prolog interpreter is XSB [14]. XSB runs on a number of platforms and has a range of interfaces. The applica-

1 Satisfaction of the ACA condition reduces the amount of work required to recover from the effects of failed transactions.

IEEE COMPUTER SOCIETY

tion programs are all written in C and use the XSB object module to produce applications offering good performance.

The C to XSB interface offers a choice of passing XSB the query in the form of a string, or building it within XSB's internal registers. As the latter method is difficult to use for all but the simplest queries, we used the string method.[2] Returned data is obtained from an XSB register.

Our method has been to present the user with the opportunity to enter a query at a web-site in the form of a schedule expressed as a sequence of Prolog predicates. In its basic format, this information is entered into a dialog box in an HTML form as if the user were presented with a Prolog prompt. However, we have also experimented with a form of interface that allows different types of operation to be selected, and where only the data item and the operation had to be entered.

Once a schedule has been entered and DTST has evaluated it, additional queries are possible (see Section III). A basic selection is available via a menu, or ad hoc queries can be entered in the form of a standard Prolog predicate.[3] Moreover, there are also several pages of on-line help available in HTML format, and these can be accessed alongside the DTST program.

The general process followed by our applications is as follows:

1. They are called by the CGI server and passed a string from which they extract the user's schedule or query.
2. The user's schedule or query is parsed to make sure that it is syntactically correct. If not, an error message is returned.
3. Where necessary, the user query is incorporated into a suitable Prolog query that will be passed to XSB for evaluation.
4. XSB is initialized and reads its pre-compiled data file; it is then passed the constructed query.
5. The result of evaluating the query is returned directly to the CGI server (embedded, of course, in the necessary HTML, as with all data returned to the server).

A particular advantage to this method is that the interface is independent of the underlying application. It is possible to build interfaces of varying levels of sophistication that can either be web-based or local applications.

We have used C for our code because of the excellent C library interface that XSB provides. However, there are a variety of alternative approaches. Of particular interest in the area of web-enabled applications is Java; the YAJXB package [7] enables XSB applications to be accessed from a Java environment. Conversely, the loose coupling of application and interface means that the system could easily be extended to accommodate additional applications. This would allow a wider selection of DBMS properties to be incorporated, and thus would make DTST a broader-based learning environment (see also Section VII).

2 The command or query to be passed to XSB is constructed within a normal C string buffer, and then passed to XSB via the interface mechanism; XSB treats it as it would treat a command entered via *stdin*.

3 Ultimately, we expect to develop a more advanced interface that allows all data entry to be input via drop-down menus.

5. Evaluating DTST

In overview, we have adopted a two-phase approach to evaluate DTST. That is, we have used a formative evaluation of DTST during the development of the learning tool. Thereafter, we conducted a summative evaluation of a prototype version of DTST.

5.1. The Formative Evaluation

For the formative evaluation of DTST a formal verification of the technically important soundness and completeness properties [10] of the software was initially performed. Thereafter, comments on the software were sought from: three members of the teaching staff at the University of Westminster (the "expert reviewers" [15]); a volunteer student from the university's MSc course in Database Systems (the one-to-one study); and a group of six volunteer students from the same course (the small-group testing). The volunteer students were randomly allocated to either the one-to-one or small-group testing (but not both). These students were learning about the CRAS properties at the time at which the formative evaluation of DTST was being conducted.

Initial demonstrations to the three expert reviewers provided some suggestions on how DTST could be improved. In particular, a number of recommendations were made on improving the user interface. The suggested improvements were made to DTST prior to us conducting the one-to-one evaluation.

The one-to-one evaluation took place over a period of two weeks and involved approximately 3 hours of contact time with the student volunteer. At the first of the one-to-one sessions, the student was introduced to DTST using a 10 minute presentation. He was provided with a quick reference guide to remind him of the basic functions supported in the version of DTST he was to use. The student was assured about the confidentiality of any information he might provide and the purpose of conducting the study was explained to him. The student was also encouraged to ask questions about DTST if he felt he needed to. Thereafter, the student was free to use DTST to explore CRAS property satisfaction using schedules of his own choice.

In the one-to-one testing, our data were gathered using observation and informal "interviews". This involved one

IEEE COMPUTER SOCIETY

of the authors sitting alongside the subject and encouraging him to articulate his feelings about the learning package as he was using it. The student reported that DTST was useful in terms of helping to develop his understanding of CRAS property satisfaction. He emphasized that DTST was motivating to used and he repeatedly suggested that the scope DTST provided, to enable him to investigate schedules of his own choosing, was important in developing understanding. The student also suggested some useful modifications to DTST. We chose to make several of the suggested changes before starting our small-group testing.

The small–group testing was performed over a four week period (approximately 8 hours of contact time spread over six sessions) with a set of six student volunteers (three males and three females). The methods employed to gather data were the same as those used for the one-to-one sessions.

Again, the power DTST provides to enable students to investigate any schedule and CRAS property was reported to be an attraction of the learning tool, and important in helping students to develop their understanding of schedule properties. The students also commented positively on the internet availability of the software: although our small-group testing was limited to that described above, all of the students involved in the formative evaluation of DTST reported that they had used the learning tool, via the web, on several additional occasions outside of the test sessions.

The feedback collected from students engaging in the small–group testing was used by us to develop DTST further and prior to its summative evaluation.

5.2. Summative Evaluation of DTST

The field test of DTST was conducted with the cohort of 29 students at the University of Westminster who were taking the Database Administration (DBA) module as part of their part-time BSc Computer Science degree programme in the Second Semester of the 2001/2002 academic year.

Students were introduced to the version of the DTST software to be summatively evaluated during a 2 hour tutorial session; the introduction was presented identically to that used in the one-to-one and small-group testing. Following the introductory session, DTST was used for the next three weeks during the part-time students' tutorial time.

In overview, there were two parts to the summative evaluation of DTST. Firstly, a t-test was performed on the results of a phase test that included questions on the CRAS properties. The t-test was intended to compare the performance of the part-time students (the experimental group) with that of the 47 full-time DBA students (the control group) who had used the standard module text [3], but not DTST. The full-time students had taken the same test in the semester before the experimental group. Secondly, a 5-point Likert scale was used to produce data on the perceptions the part-time students had of DTST and [3], as methods for facilitating understanding of the CRAS properties, and their attractiveness as learning instruments. A t-test was again used to analyse the data produced and was based on a comparison of the matched pairs of scores produced by each respondent for DTST and [3].

Mean test scores for the part-time and full-time students for the phase test were calculated for both the CRAS and non-CRAS related questions to compare the performance of the two sets of students. Because we felt that DTST might have had an effect in encouraging learning gains, we chose to test the alternative (directional) hypothesis that: the part-time students performed better on the test of CRAS property understanding than the full-time students. The corresponding null hypothesis was that there was no difference in the phase test scores produced by the two sets of students.

The analysis of our data showed that the mean test scores (out of 12) for the full-time and part-time students on the questions on CRAS properties were 6.52 and 8.38 respectively (the corresponding standard deviations were 9.65 and 6.97, respectively). The computed t-statistic was 0.81 for 27 degrees of freedom. Hence, the directional hypothesis had to be rejected in favour of the null hypothesis.

A comparison of the mean scores (out of 28) achieved by the students on that part of the phase test that did not directly relate to the CRAS properties revealed that the average mark for full-time students was 16.96 whereas the average mark for part-time students was 17.58. As such, whereas the average score for the full-time students on the phase test questions relating to the CRAS properties was 29% lower than the part-time students, the average mark for the full-timers on questions not related to the CRAS properties was only 4% lower. Although these figures do not prove anything, they offer some *suggestive* evidence that DTST might have helped students to understanding the CRAS properties.

The combination of the statistically non-significant analysis of the phase test scores and the fact that a number of potentially confounding variables applied in our study meant that we were not able to draw any firm conclusions on whether DTST had been of value in terms of helping students understand the CRAS properties.

The Likert scale included a total of 24 statements (with an equal number of positive and negative statements). These items were divided into three categories. Eight of the statements were intended to measure the extent to which DTST and [3] were perceived as being of value in facilitating student understanding of the CRAS properties, a further eight items were intended to help to decide the extent to which DTST and [3] were motivating to use, and the remaining eight statements were used to collect the students' opinions on the value of comparable features of DTST and [3] (i.e., their explanations, exercises and examples). Students were

IEEE COMPUTER SOCIETY

asked to indicate their strength of agreement/disagreement with each statement in the Likert scale. The five options were: strongly agree, agree, unsure, disagree and strongly disagree.

26 responses to the Likert scale were returned. To produce the measures of student attitudes, a three-stage approach was adopted. The initial step involved "signing" the 24 items included in the Likert scale as being a positive or negative statement about DTST or [3]. Next, the returns were analysed using the following system: for each positive statement a response of "strongly agree" was given a score of 5, an "agree" response was given a score of 4, a score of 3 corresponded to an "undecided" response, "disagree" was scored as a 2, and "strongly disagree" was scored as a 1. Conversely, for each negative statement, a "strongly agree" response was given a score of -5, "agree"was scored as -4, "undecided" was recorded as a -3, "disagree" was given a score of -2, and -1 corresponded to a "strongly agree" response. By summing the scores for each return, a figure corresponding to the respondent's attitude towards DTST and [3] was computed. In the final step, the [3] score for each respondent was subtracted from the score for DTST. This calculation gave a measure of a respondent's attitude to DTST that is relative to their attitude towards [3].[4]

To analyse the information produced from the Likert scale, t-statistics were computed to compare the mean scores for the perceptions students had of DTST and [3], overall and for each of the three categories of items included in the Likert scale.

In the overall measure of the two methods, the average difference in the ratings of DTST and [3] was 16.79 in favour of DTST, and no student reported that [3] was "better" than DTST. The t-statistic for the comparison of average differences was 2.25. This is statistically significant at the 2% level. Not surprisingly, given the overall results, DTST was also perceived to be "better" than [3] in all three of the sub-categories of Likert scale items.

In terms of facilitating understanding of the CRAS properties, the average difference in scores between DTST and [3] was 2.07, in favour of DTST, and all but four of the students reported that DTST had been more valuable than [3] on this measure. In the t-test comparison of the average difference in the ratings of DTST and [3], the t-statistic was 2.18. This value is significant at the 5% level.

The average difference in the rating of DTST and [3] on motivational appeal was 10.85, and all but one student reported that DTST had been more motivating to use than [3]. The t-value, of 2.51, for the comparison of average differences in ratings between DTST and [3] on motivational appeal is significant at the 2% level.

4 A positive score indicates a more favourable attitude towards DTST than [3]; a negative score represents a more favourable attitude towards [3] than DTST.

The average difference in scores on the value of the exercises, explanations and examples was 4.63 in favour of DTST. The t-statistic for the average difference was 3.21, which is significant at the 1% level.

6. Conclusions and Further Work

The results of our analysis of DTST suggest that the tool is of value to students learning about the CRAS properties. In future developments of DTST, we aim to support other types of schedule properties (e.g. *rigour* [5]).

Although currently focussed on the CRAS properties, extended forms of DTST are possible to support students' learning other topics in the University-level Computer Science curriculum (e.g., database recovery techniques and *state machines* [8]). Being web-based, DTST is suitable for use by Computer Science students taking courses in distance learning mode; investigating the use of DTST in a distance learning environment is a matter for further work.

References

[1] *The Apache Software Foundation*. http://www.apache.org.

[2] S. Barker. Proving properties of schedules. In *Proc. IEEE Workshop on Knowledge and Data Engineering*, pages 174–180, 1998.

[3] P. Bernstein, N. Goodman, and V. Hadzilacos. *Concurrency Control and Recovery in Database Systems*. Addison Wesley, 1987.

[4] I. Bratko. *PROLOG Programming for Artificial Intelligence*. Addison-Wesley, 1986.

[5] Y. Briebart, D. Georgakopoulos, M. Rusinkiewicz, and A. Silbershatz. On rigorous transaction scheduling. In *IEEE Transactions on Software Engineering*, 17, pages 954–960, 1991.

[6] *CGIHTML*. http://www.eekim.com/software/cgihtml/.

[7] S. Decker. Yajxb website. http://www-db.stanford.edu/

[8] Y. G. et al, editor. *Abstract State Machines - Theory and Applications*. Springer, 2000.

[9] R. M. Gagne. *The Conditions of Learning*. Holt, Reinhart and Winston, 1970.

[10] M. Genesereth and N. Nilsson. *Logical Foundations of Artificial Intelligence*. Morgan Kaufmann, 1987.

[11] F. Marton and P. Ramsden. *What does it take to improve learning?* Kogan Page, 1988.

[12] *NCSA: The Common Gateway Interface*. http://hoohoo.ncsa.uiuc.edu/cgi/.

[13] J. Nichol, J. Briggs, and J. Dean. *Prolog, Children and Students*. Kogan-Page, 1988.

[14] K. Sagonas, T. Swift, D. Warren, J. Freire, and P. Rao. *The XSB System Version 2.0, Programmer's Manual*, 1999.

[15] M. Tessmer. *Planning and Conducting Formative Evaluations*. Kogan-Page, 1993.

IEEE COMPUTER SOCIETY