# An e-learning tool for database administration.

**Paul Douglas**
Cavendish School of Computer Science, University of Westminster

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

# An E-learning tool for Database Administration

Paul Douglas
University of Westminster, London, UK
P.Douglas@wmin.ac.uk

Steve Barker
King's College, London, UK
steve@dcs.kcl.ac.uk

## Abstract

*In this paper, we describe an item of "intelligent" educational software that is intended to help students taking university computer science courses to understand the fundamentals of transaction scheduling. The software, implemented in PROLOG, empowers students to construct their own learning environment and is able to provide tailored forms of feedback to different types of learner. We describe the development and evaluation of the software, and we present details of the analysis of the results of our investigation into the effectiveness of the software as a teaching and learning tool. Our results suggest that our learning tool provides students with a different and valuable type of learning experience, which traditional methods do not provide.*

**Keywords**
E-Learning, Educational Software, Databases, Transaction Scheduling, Recovery.

## 1 Introduction

In this paper, we describe an item of educational software that we have developed and used to help us to teach certain key notions from the realms of database transaction processing to undergraduate computer science students. More specifically, the software is an educational tool that is intended to "intelligently" assist computer science students in developing their understanding of schedule property satisfaction. In this context, "intelligently" may be interpreted as an ability to respond to a student's self-selected input by detecting and explaining his/her errors to them or confirming that his/her understanding is correct.

Ours is one of the first pieces of courseware to provide students with help in understanding the basic notions of database transaction processing and, to the best of our knowledge, is the first piece of software that is specifically intended for helping students to learn about properties of schedules. The software provides students with a tutorial aid that is able to respond to questions about schedule property satisfaction in the same way that an "expert tutor" might; it enables a student to investigate schedule properties at his/her leisure and enables teaching staff to use tutorial sessions to answer any "non-standard" questions that students might have. This tool is also important because it provides students with a learning experience that no textbook can provide. More specifically, the software encourages students to learn about schedule properties by making and testing hypotheses. This approach appears to be the most natural way for students to learn about schedule properties (certainly it is the approach they naturally adopt). The traditional, text-based method that we have previously used to teach material on schedule property satisfaction does not support learning by hypothesis formulation.

The rest of the paper is organized in the following way. Section 2 provides a brief introduction to schedule properties. In Section 3, some key features of the software are outlined. In Section 4, the main results produced from the evaluation of the software are described and discussed. In Section 5, some conclusions are drawn, and suggestions are made for further work.

## 2 Schedule Properties

Schedule properties are criteria which should be satisfied by a *schedule*, a sequence of interleaved read and write operations performed on the objects which are



Proceedings of the International Conference on Information Technology: Coding and Computing (ITCC'05)
0-7695-2315-3/05 $ 20.00 **IEEE**

stored in a database. These operations are performed by a database management system (DBMS) on behalf of a set of transactions on which the schedule is defined.

Unfortunately, certain interleavings of the operations from different transactions in a schedule can cause anomalous behaviours (which cause the integrity of the data in a database to be compromised) and can raise a number of practical difficulties. For instance, it is possible for the value of a data item which has been updated by one transaction $t_i$ in a schedule to be overwritten by another transaction $t_j$ before $t_i$'s update is performed on the database (this phenomenon is usually referred to as the *lost update problem* [5]).

Schedule properties [6] solve this type of problem by imposing certain constraints on the order in which operations are performed in a schedule; by ensuring that these constraints are satisfied, a database management system is guaranteed to produce schedules which are free of a class of potential problems which may violate the integrity of the data contained in a database. Moreover, the DBMS can be configured to optimize the performance of transaction processing.

The schedule properties that we have implemented are: conflict serializability, view serializability, recoverability, avoids cascading aborts, strictness, and rigor. These properties are defined formally below. In these definitions, $t_i$ and $t_j$ denote arbitrary transactions, $T(\sigma)$ denotes an arbitrary schedule $\sigma$ defined on a set of transactions $T$, $r_i$, $w_i$, $a_i$ and $c_i$ are respectively read, write, abort and commit operations by transaction $t_i$, $\rightarrow$ is "implication", $\wedge$ is 'and', $\vee$ is 'or', $\neg$ is negation, and $<$ denotes the "earlier than" relationship between operations.

**Definition 2.1** *A schedule $\sigma$ on a set of transactions $T$ is conflict serializable iff the following holds:*

$$\forall t_i, t_j \in T(\sigma)\ conflict(t_i, t_j) \rightarrow \neg conflict(t_j, t_i)$$

*where conflict is defined thus:*

$$\forall t_i, t_j \in T(\sigma)\ r_i(x) < w_j(x) \rightarrow conflict(t_i, t_j)$$

$$\forall t_i, t_j \in T(\sigma)\ w_j(x) < r_i(x) \rightarrow conflict(t_j, t_i)$$

$$\forall t_i, t_j \in T(\sigma)\ w_i(x) < w_j(x) \rightarrow conflict(t_i, t_j).$$

**Definition 2.2** *A schedule $\sigma$ on a set of transactions $T$ is recoverable iff the following holds:*

$$\forall t_i, t_j \in T(\sigma)\ read\_from(t_i, t_j) \rightarrow c_j \in \sigma\ \wedge\ c_j < c_i.$$

**Definition 2.3** *A schedule $\sigma$ on a set of transactions $T$ avoids cascading aborts iff the following holds:*

$$\forall t_i, t_j \in T(\sigma)\ read\_from(t_i, t_j) \rightarrow c_j < r_i(x) \vee a_j < r_i(x).$$

**Definition 2.4** *A schedule $\sigma$ on a set of transactions $T$ is strict iff the following holds:*

$$\forall t_i, t_j \in T(\sigma)\ w_j(x) < r_i(x) \vee w_j(x) < w_i(x) \rightarrow a_j < r_i(x) \vee c_j < r_i(x) \vee a_j < w_i(x) \vee c_j < w_i(x).$$

**Definition 2.5** *A schedule $\sigma$ on a set of transactions $T$ satisfies the property of rigour iff $\sigma$ is strict and the following holds:*

$$\forall t_i, t_j \in T(\sigma)\ r_j(x) < w_i(x) \rightarrow a_j < w_i(x) \vee c_j < w_i(x).$$

**Definition 2.6** *The auxiliary predicate read_from is defined thus:*

$$\forall t_i, t_j \in T(\sigma)\ \exists x[read\_from(t_i, t_j) \leftarrow w_j(x) < r_i(x) \wedge \neg(a_j < r_i(x)) \wedge [\forall t_k \in T(\sigma)\ w_j(x) < w_k(x) < r_i(x) \rightarrow a_k < r_i(x)]].$$

**Definition 2.7** *For each data item $x$, if (i) $w_i(x) < r_j(x)$ holds in a schedule, (ii) $t_i$ does not abort before $r_j(x)$, and (iii) every transaction (if any) that writes $x$ between $w_i(x)$ and $r_j(x)$ aborts before $r_j(x)$ then $t_j$ reads from $t_i$. In other words, $r_j(x)$ is a read from $w_i(x)$ if $w_i(x)$ is the last transaction to have written $x$ prior to $r_j(x)$, and $t_i$ has not aborted.*

**Definition 2.8** *For each data item $x$, $w_i(x)$ is a final write of $x$ in a schedule $\sigma$ if $t_i$ commits and there is no $w_j(x)$ operation in $\sigma$ such that $w_i(x)$ precedes $w_j(x)$ and $t_j$ commits.*

**Definition 2.9** *Two schedules, $\sigma_1$ and $\sigma_2$, are view equivalent iff the following conditions are satisfied:*

1. *$\sigma_1$ and $\sigma_2$ include the same set of committed transactions and the same sets of operations;*

2. *$t_j$ reads $x$ from $t_i$ in $\sigma_1$ iff $t_j$ reads $x$ from $t_i$ in $\sigma_2$;*

3. *For each data item $x$, the transaction that performs the final write of $x$ in $\sigma_1$ must perform the final write of $x$ in $\sigma_2$.*

**Definition 2.10** *A schedule $\sigma$ over a set of transactions $\mathcal{T}$ is view serializable if the committed projection of $\sigma$ is view equivalent to some serial schedule over $\mathcal{T}$.*

## 3 An Overview of the Software

Our teaching tool enables students to test any syntactically correct schedule that they choose as input to the system. Students also have complete freedom to choose to investigate the satisfaction of any of schedule properties by these schedules.

The software which implements the system is written in PROLOG. PROLOG has been widely used for implementing items of educational software (see, for example, [10] and [8]) and is appropriate for developing applications, like this, which require that some form of "intelligence" be captured [2]. The fact that the rules which define schedule properties can be directly translated into PROLOG's rule-based language was another reason for choosing the latter for the implementation of the software.

Our design of the software has been influenced by Gagne's work [3]. Gagne's event-based model of instruction helped us to decide what an individual learner ought to be offered and the order in which information ought to be presented to them. Following Gagne's suggestions, when students use the software 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. Prior to developing our software we adopted a *phenomenographic* method [7] 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 schedule 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 schedule property satisfaction.

When engaging with our software, a user enters a schedule and selects a schedule property to evaluate with respect to the schedule. The schedule is displayed to the user who may then pose queries on the schedule to test it for satisfaction of schedule properties with respect to the set of axioms $\mathcal{A}$ that defines these properties and the auxiliary predicates in terms of which schedule properties are defined (see Definitions 2.4-2.8). The axioms in $\mathcal{A}$ are converted into PROLOG code for implementation.

Each operation in a schedule may be represented by a 4-tuple, ($o$,$t_j$,$i$,$t_s$). Here, $o$ denotes an operation (i.e. read or write), $t_j$ denotes a transaction performing the operation, $i$ denotes the data item read or written by $t_j$, and $t_s$ is the time at which $o$ is performed i.e., the timestamp for $o$. In the case where $o$ is a commit or an abort, the data item is *null* since these operations are not performed on a data item. In our PROLOG implementation, each 4-tuple that describes an operation is represented as a fact of the form $o(a, t_j, i, t_s)$ where $a \in \{r, w\}$. In this context, a schedule $\sigma$ is a finite set of $o$ operations, and a PROLOG program is a pair $(\mathcal{A}^P, \sigma)$ where $\mathcal{A}^P$ is the PROLOG form of $\mathcal{A}$.

An example of the output for a schedule $\sigma$ produced in a user session and an example of engaging with the system follows next.

**Example 3.1** *Suppose that a user's choice of schedule is as follows:*

$$\langle w_1(x), w_1(y), w_1(z), w_2(z), c_1, w_2(x), r_2(y), w_2(y), c_2 \rangle$$

*Then, the software displays the user schedule, thus:*

*Your chosen schedule was:*

*w,1,x,90*
*w,1,y,95*
*w,1,z,105*

*w,2,z,110*
*c,1,null,115*
*w,2,x,120*
*r,2,y,125*
*w,2,y,130*
*c,2,null,135*

*Thereafter, the user may evaluate schedule properties with respect to the schedule. For example, the user may ask "is this schedule an ACA schedule?" i.e., ?-aca.. In this case the output is "yes". A user can ask for an explanation of this result by posing the following query: ?-explainaca. To which the software will respond:*

> *Transactions in this schedule only read data items AFTER they have been written by transactions that have committed.*

*Similarly, the query ?-st. for the schedule above (i.e. "is this schedule strict?") is answered "no" by the software. If the user then poses the query ?-explainnonst. (i.e. why is this schedule not strict?) then the software will respond with the following explanation:*

> *Transaction 2 overwrites the data item z written by transaction 1 but BEFORE transaction 1 reaches its commit point.*

All schedule property satisfaction questions are evaluated as described in the previous example, and several levels of explanation are provided by the software.[1]

Our early development and testing made use of the PROLOG program run by the students on the University of Westminster Unix system.

Although students had no real difficulty using the command-line-driven interface that was initially developed, continued evaluation of the software revealed that students found entering the schedule and testing queries frustrating. For that, students had to enter complete schedules into a file and had to recompile the application each time a new schedule was chosen to test some schedule property. In response, we have developed a prototype web-based interface for the program:

and providing the user with feedback:

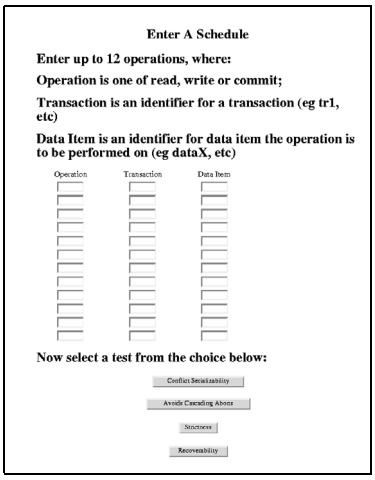


**Figure 1. Entering a Schedule**

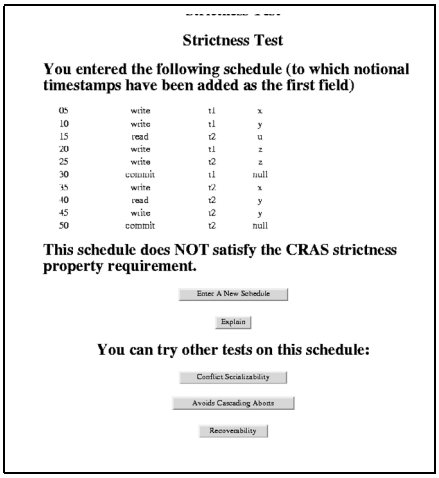


**Figure 2. Program Response**

## 4 Testing the Software

We carried out primarily formative testing of the software, with some additional summative testing to evaluate student perceptions of using the software. We focussed primarily on testing the part of the software aimed at teaching schedule properties (students tend to find this difficult).

In overview, for the formative evaluation of the software a formal verification of the technically important

[1] Several levels of explanation are offered in the sense that users are able to check the correctness of a query in respect of any of schedule properties. If the query is not correct they can attempt to correct it; alternatively, they can ask the software *why* it is not correct.

soundness and completeness properties [4] of the software was initially performed. Thereafter, comments on the software were sought from: two members of the teaching staff at the University of Westminster (the "expert reviewers", Tessmer, 1993); a volunteer student from the university's MSc course in Software Engineerings (the one-to-one study); and a group of four volunteer students from the same course (the small-group testing).

Initial demonstrations to the two expert reviewers provided some suggestions on how the software could be improved. In response, the explanations of schedule property violation were changed to try to make them more meaningful.

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 the software using a 10 minute presentation, and was provided with a quick reference guide to remind her of the basic functions supported in the version of the software she was to use. We deliberately chose to make the introduction short in order to see whether the software was as easy to use as we anticipated it would be. The student was assured about the confidentiality of any information she might provide and the purpose of conducting the study was explained to her. She was also encouraged to ask questions about the software if she felt she needed to. Thereafter she was free to use the software on her own.

In the one-to-one test our data was gathered using observation and informal "interviews". This involved one of us sitting alongside the subject and encouraging her to articulate her feelings about the learning package as she was using it. In general, the student was quite enthusiastic about the software and reported that it was of value in helping to improve her understanding of schedule property satisfaction. She particularly liked the fact that the software enabled her to decide what schedules and what schedule properties she wanted to investigate. She suggested that this was a major advantage of the software relative to a text, like [1], which can offer only a fixed set of predetermined schedules. She also suggested a number of useful modifications which we chose to make.

The small-group testing was performed over a three week period (involving approximately 4 hours of contact time spread over five sessions) with the set of four student volunteers. The main purpose of the small-group evaluation was to collect data by observing a larger group of students using the software. The methods employed to gather this data were the same as those used for the one-to-one sessions. As with the one-to-one sessions, the power to investigate any schedule and schedule property was reported to be an attraction of the software; the students commented that they particularly liked the fact that they could "interact" with the software, and that it "lets you decide what to learn".

After incorporating some minor modifications that were suggested by the students involved in the small-group testing, we introduced to a full cohort (18) of MSc students. The students used the software during tutorials for a three week period, and were encouraged to use it further for individual work.

In order to attempt to minimize the possibility of any researcher bias in measuring student attitudes, we conducted our summative testing by using a Likert scale to collect data about the perceptions the students had of the software and [1] as methods for facilitating understanding of schedule properties, and their attractiveness as learning instruments. To analyse the data produced from the Likert scale, we chose to use a t-test; the idea was to compare the matched pairs of scores produced by each respondent for the software and [1].

A 5-point Likert scale was used for the investigation of student perceptions of the value of the software and [1], in terms of helping them to learn about schedule properties, the motivational appeal of the different forms of instruction and the perceived relative value of some common features of the software and [1] which could be compared.

To analyze the information produced from the Likert scale, t-statistics were computed to compare the mean scores for the perceptions students had of the software and [1]. The results produced from the Likert scale were very clear. In the overall measure of the two methods, the average difference in the ratings of the software and [1] was 17.24 in favour of the software, and no students reported that [1] was "better" than the software. The t-statistic for the comparison of average differences was 7.75. This is statistically significant at the 1% level.

It would be interesting in future to try to compare the assessment results of students who have used our learning tool with the results of those who have not. However, this would ideally involve comparing the results of two groups of students taking the same module at the same time, and various ethical issues arise when attempting to undertake this kind of test. It might be possible to compare the results achieved by a group of students who use the software in a future year with those of past students who did not.

## 5 Conclusions and Further Work

Our software shows that a suitable tool can be developed to help computer science students to learn about schedule properties. The package enables students to construct their own learning environments by using a piece of courseware that is able to interpret and immediately explain a student's mistakes as well as being able to confirm it when his/her understanding is correct. As such, the software provides students with "intelligent" tutorial support for learning about schedule properties. The software is also based on sound principles of learning [1], is able to deal with any syntactically correct schedule, and it can be extended to accommodate any number of examples or exercises without requiring changes to the core set of rules on which the software is based.

Using the software enables students to: choose to investigate any of the schedule properties using schedules of their own choice; make hypotheses about schedules satisfying schedule properties; test these hypotheses; and explore the consequences of schedule property satisfaction by manipulating the operations included in a schedule. As such, the software empowers students to take control of their own learning, they can learn at their own preferred pace, they can investigate their own misunderstandings and reinforce their own understanding of schedule properties. The fact that the software encourages students to "learn by hypothesizing" is particularly important because this is the approach students naturally adopt to learn about schedule properties. Using textbooks does not enable this type of learning to be supported, and can only offer students a limited number of schedules and examples of schedule property satisfaction; textbooks cannot provide interactive feedback to students investigating schedules and schedule properties of their own choosing. Unlike their human tutors, the software has the additional attraction of providing students with tutorial support in their learning of the schedule properties whenever they require it.

The results produced by our summative assessment of the software indicate that it was perceived by our students to be superior to [1] in a number of respects. However, more work will be required on the issue of student perceptions of the software and [1] before any definite conclusions may be drawn about their relative value. Our experience of conducting this study has also revealed that some students have a tendency to believe that a piece of educational software has to invariably be better than a text; these students regard the former as being "the future" whilst the latter is viewed as being distinctly *passé*. We intend to investigate the implications of this attitude in the near future. Moreover, while the Likert scale test revealed that the software was perceived to be helpful to students learning about schedule properties, further research is required to try to establish *why* this is the case.

A number of extensions to the software are possible. For example, with minor modifications the software can be used as a tutorial aid for learning about *optimistic concurrency control* and various locking protocols may be implemented.

## References


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

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

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

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

[5] J. Gray and A. Reuter. *Transaction Processing: Concepts and Techniques*. Morgan Kaufmann, 1993.

[6] V. Kumar. *Performance of Concurrency Control Mechanisms in Centralised Database Systems*. Prentice Hall, 1996.

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

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

[9] M. Yazdani. *New Horizons in Educational Computing*. Ellis Horwood, 1983.