# A Generic Tool to Browse Tutor-Student Interactions: Time Will Tell!

Jack Mostow, Joseph Beck, Andrew Cuneo, Evandro Gouvea, and Cecily Heiner
*Project LISTEN (www.cs.cmu.edu/~listen), Carnegie Mellon University*
*RI-NSH 4213, 5000 Forbes Avenue, Pittsburgh, PA.  USA 15213-3890*

**Abstract.** A basic question in mining data from an intelligent tutoring system is, "What happened when…?" A generic tool to answer such questions should let the user specify which phenomenon to explore; explore selected events and the context in which they occurred; and require minimal effort to adapt the tool to new versions, to new users, or to other tutors. We describe an implemented tool and how it meets these requirements. The tool applies to MySQL databases whose representation of tutorial events includes student, computer, start time, and end time. It infers the implicit hierarchical structure of tutorial interaction so humans can browse it. A companion paper [1] illustrates the use of this tool to explore data from Project LISTEN's automated Reading Tutor.

## 1. Introduction

Intelligent tutoring systems' ability to log their interactions with students poses both an opportunity and a challenge. Compared to human observation of live or videotaped tutoring, such logs can be more extensive in the number of students, more comprehensive in the number of sessions, and more exquisite in the level of detail. They avoid observer effects, cost less to obtain, and are easier to analyze. For example, Project LISTEN's Reading Tutor listens to children read aloud, and helps them learn to read [2]. A Reading Tutor session consists of reading a number of stories. The Reading Tutor displays a story one sentence at a time, and records the child's utterances for each sentence. The Reading Tutor logs each event (session, story, sentence, utterance, …) into a database table for that event type. Data from tutors at different schools flows into an aggregated MySQL database server [3]. Our 2003-2004 database includes 54,138 sessions, 162,031 story readings, 1,634,660 sentences, 3,555,487 utterances, and 10,575,571 words. Such data is a potential gold mine [4].

Mining such data requires the right tools to locate promising areas, obtain samples, and analyze them. One part of this process is in-depth qualitative analysis of individual tutorial events. Such case analyses serve various purposes, for example:

- Spot-check tutoring sessions to discover undesirable tutor-student interactions.
- Identify the most common types of cases in which a specified phenomenon occurs.
- Formulate hypotheses by identifying features that examples suggest are relevant.
- Sanity-check a hypothesis by checking that it covers the intended sorts of examples.

This paper describes a tool, implemented as a Java™ program that queries MySQL databases, that supports such case analysis by exploiting three simple but powerful ideas. First, a student, computer, and time interval suffice to specify an event. Second, a containment relation between time intervals defines a hierarchical structure of tutorial interactions. Third, the first two ideas make it possible to implement a generic but flexible tool for mining tutor data with minimal dependency on tutor-specific details.

## 2. Specify which phenomenon to explore.

First, how can we specify events to explore? A deployed tutor collects too much data to look at, so the first step in mining it is to select a sample. A database query language provides the power and flexibility to describe and efficiently locate phenomena of interest.

For example, the query "`select * from utterance order by rand() limit 10`" selects a random sample of 10 from the table of student utterances. Whether the task is to spot-check for bugs, identify common cases, formulate hypotheses, or check their sanity, our mantra is "check (at least) ten random examples." Random selection assures variety and avoids the sample bias of, for example, picking the first ten examples in the database.

Although an arbitrary sample like this one is often informative, a query can focus on a particular phenomenon of interest, such as: Which questions did students take longest to answer? Or: When did students get stuck long enough for the Reading Tutor to prompt them? Exploring examples of such phenomena can help the researcher spot common features and formulate causal hypotheses to test with statistical methods on aggregated data.

Second, what information suffices to identify a tutorial interaction? A key insight here is that student, computer, and time interval are sufficient, because together they uniquely specify the student's interaction with the tutor during that time interval. (We include computer ID in case the student ID is not unique.) This "lowest common denominator" should apply universally to virtually any tutor.

Third, how can we translate the result of a query into a set of tutorial events? The tool scans the labels returned as part of the query, and finds the columns for student, computer, start time, and end time. The code assumes particular names for these columns, e.g. "`user_id`" for student, "`machine_name`" for computer, and "`start_time`" for start time. If necessary the user can enforce this naming convention, e.g., by inserting "`as start_time`" in the query to relabel the column. We require that the fields for student, computer, start time, and end time be keys in the database tables. Indexing tables on these fields enables fast response by the tool even for tables with millions of records.

## 3. Explore selected events and the context in which they occurred.

What context frames an event? Our answer is: "its chain of ancestor events." *E.g.*, the context of a word includes the utterance, sentence, story, and session in which it was read.

How can we discern the hierarchical structure of student-tutor interaction? At first we computed this hierarchy using its hardwired schema for the Reading Tutor database to determine which events are part of which others. But then we had a key insight: exploit the nested time intervals in the data logged by our tutor – and probably by many others too.

If events A and B have the same student and computer, when is A an ancestor of B? We initially required that A contain all of B. But we relaxed the criterion to better handle occasional overlapping intervals in our data. We therefore define A as an ancestor of B if B starts during A. Thus a word's ancestors include an utterance, sentence, story, and session.

The tool computes the event tree by partial-ordering the events according to the ancestor relation. The parent of an event is defined as its minimal ancestor. Siblings are defined as sharing the same parent; they are ordered by their start times.

The companion paper [1] shows how the tool displays such trees, summarizes events in readable form, and lets users dynamically drill down and adjust which details to display.

## 4. Require minimal effort to adapt the tool to new versions, new users, or other tutors.

How can the tool obtain the information it needs about a database of tutor interactions? Its generic architecture enables it to make do with readily available meta-data, a few assumed conventions, and a little code. MySQL provides the required meta-data, namely the list of tables in the database, the fields in each table and event list, and their names and data types. We exploit the observation (or assumption) that the meaning of field names is consistent across database tables and over time. The code assumes particular field names for student, machine, and start and end times, but overrides this convention when necessary, as in the case of a particular table with a "`Time`" field instead of a "`Start_time`" field.

The method to compute the context of a selected target event is: First, extract its student, computer, and start time. Then query every table of the database for records for the same student and computer whose time interval contains the start of the target event. Finally, sort the retrieved records according to the ancestor relation, and display them accordingly by inserting them at appropriate positions in a Java™ expandable tree widget.

The method to find the children of a given event fires only when needed to expand the event node. It finds descendants in much the same way as the method to find ancestors, but then winnows them down to the children (those that are not descendants of others). Both methods work whether the events are in the same table or in different tables.

A more knowledge-based method would know which types of Reading Tutor events can be parents of which others. However, this knowledge would be tutor- and possibly version-specific. In contrast, our brute force solution of querying all tables requires no such knowledge. Moreover, its extra computation is not a problem in practice. Our databases consist of a few dozen tables, the largest of which have tens of millions of records. Despite this table size, the tool typically computes the context of an event with little or no delay.

## 5. Conclusion

This paper reports an implemented, efficient, generic solution to a major emerging problem in educational data mining: efficient exploration of vast student-tutor interaction logs. We describe three useful requirements for such exploration that an earlier tool [5] failed to meet, and how the new tool meets them: let the user specify which phenomenon to explore; explore selected events and the context in which they occurred; and require minimal effort to adapt the tool to new versions, to new users, or to other tutors. Our key conceptual contribution uses temporal relations to expose natural hierarchical structure. This is the sense in which "time will tell" many basic relationships among tutorial events.

The success of this approach suggests specific recommendations in designing databases of tutorial interactions: Log each distinct type of tutorial event in its own table. Include student ID, computer, start time, and end time as fields of each such table so as to identify its records as events. Name these fields consistently within and across databases created by successive versions of the tutor so as to make them easier to extract.

The ultimate test of this tool is whether it leads to useful discoveries, or at least sufficiently facilitates the process of educational data mining that the miners find it helpful and keep using it. To repeat our subtitle in its more usual sense, "time will tell!"

**Acknowledgements**: This work was supported in part by the National Science Foundation under ITR/IERI Grant No. REC-0326153. Any opinions, findings, conclusions, or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the National Science Foundation or the official policies, either expressed or implied, of the sponsors or of the United States Government. We thank the educators and students who generated our data.

**References (see www.cs.cmu.edu/~listen)**

1. Mostow, J., J. Beck, H. Cen, E. Gouvea, and C. Heiner. Interactive Demonstration of a Generic Tool to Browse Tutor-Student Interactions. *Supplemental Proceedings of the 12th International Conference on Artificial Intelligence in Education (AIED 2005)* 2005. Amsterdam.
2. Mostow, J., G. Aist, P. Burkhead, A. Corbett, A. Cuneo, S. Eitelman, C. Huang, B. Junker, M.B. Sklar, and B. Tobin. Evaluation of an automated Reading Tutor that listens: Comparison to human tutoring and classroom instruction. *Journal of Educational Computing Research*, 2003. *29*(1): p. 61-117.
3. MySQL. Online MySQL Documentation. 2004.
4. Beck, J., ed. *Proceedings of the ITS2004 Workshop on Analyzing Student-Tutor Interaction Logs to Improve Educational Outcomes*. 2004: Maceio, Brazil.
5. Mostow, J., J. Beck, R. Chalasani, A. Cuneo, and P. Jia. Viewing and Analyzing Multimodal Human-computer Tutorial Dialogue: A Database Approach. *Proceedings of the ITS 2002 Workshop on Empirical Methods for Tutorial Dialogue Systems*, 75-84. 2002. San Sebastian, Spain.