Lessons from Project LISTEN's Session Browser

Jack Mostow, Joseph Beck, Andrew Cuneo, Evandro Gouvea, Cecily Heiner, and Octavio Juarez

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 tool to answer such questions should let the user specify which phenomena to explore; find instances of them; summarize them in human-understandable form; explore the context where they occurred; dynamically drill down and adjust which details to display; support manual annotation; and require minimal effort to adapt to new tutor versions, new users, new phenomena, or other tutors.

This chapter describes the Session Browser, an educational data mining tool that supports such case analysis by exploiting three simple but powerful ideas. First, logging tutorial interaction directly to a suitably designed and indexed database instead of to log files eliminates the need to parse them and supports immediate efficient access. Second, a student, computer, and time interval together suffice to identify a tutorial event. Third, a containment relation between time intervals defines a hierarchical structure of tutorial interactions. Together, these ideas make it possible to implement a flexible, efficient tool to browse tutor data in understandable form yet with minimal dependency on tutor-specific details.

We illustrate how we have used the Session Browser with MySQL databases of millions of events logged by successive versions of Project LISTEN's Reading Tutor. We describe tasks we

have used it for, improvements made, and lessons learned in the years since the first version of the Session Browser [1-3].

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. The resulting data is a potential gold mine [4] – but mining it requires the right tools to locate promising areas, obtain samples, and analyze them.

Place near here:

Figure 1: Picking a story in Project LISTEN's Reading Tutor

Place near here:

Figure 2: Assisted reading in Project LISTEN's Reading Tutor

For example, consider the data logged by Project LISTEN's Reading Tutor [5, 6], which listens to children read aloud, and helps them learn to read [7-10]. As Figure 1 and Figure 2 illustrate, each session involves taking turns with the Reading Tutor to pick a story to read with its assistance. The Reading Tutor displays the 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 flow overnight into an aggregated database on our server. For example, 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. This data is potentially very informative, but orders of magnitude larger than is feasible to peruse in its entirety.

We view educational data mining as an iterative cycle of hypothesis formation, testing, and refinement. This cycle includes two complementary types of activities. One type of activity involves aggregate quantitative data analysis, whether confirmatory or exploratory. For example, embedded experiments [5, 11-13] compare alternative tutorial actions by selecting randomly among them and aggregating outcomes of many trials. Knowledge tracing [14] and learning decomposition [15] analyze growth curves by aggregating over successive opportunities to apply a skill. The Pittsburgh Science of Learning Center's (PSLC) DataShop [16] summarizes performance and learning curves aggregated by student, problem, problem step, and/or skill.

In contrast, the other type of activity involves in-depth qualitative analysis of individual examples. For conventional educational data in the form of test scores, this type of exploratory data analysis might try to identify individual students' patterns of correct and incorrect answers to individual test items, or (if available) what they wrote on the test paper in the course of working them out. For data logged by an intelligent tutoring system detailing its interactions with students, such analysis might try to identify significant sequences of tutorial events. In both cases, the research question addressed is descriptive [17, p. 204]: "What happened when …?" Such case analyses serve any of several purposes, for example:

- Spot-check tutoring sessions to discover undesirable tutor-student interactions.
- Identify and characterize typical cases in which a specified phenomenon occurs.
- Develop a query by refining it based on examples it retrieves.

- Formulate hypotheses by identifying features that examples suggest are relevant.
- Sanity-check a hypothesis by checking that it covers the intended sorts of examples.

1.1 Relation to prior research

An earlier program [18] attempted to support case analysis by viewing interactions logged by Project LISTEN's Reading Tutor. As Table 1 and Table 2 illustrate, the viewer displayed a table of tutorial interactions at a user-selected level of detail – computers, launches, students, sessions, stories, sentences, utterances, words, or clicks. For example, Table 1 lists stories encountered in a particular session on April 5, 2001, and Table 2 lists sentences encountered during the first story. Some columns of this table contained hyperlinks that let the user drill down to a table at the next deeper level of detail. For example, clicking on the hyperlink labelled 6 in the Sentence Encounters column of the first row of the table shown in Table 1 caused the viewer to display the table of 6 sentence encounters shown in Table 2. This ability to generate a table on demand with a single click spared the user the effort of writing the multiple database queries, including joins, required to generate the table.

Place near here:

Table 1: Activities in a session [18]

Place near here:

Table 2: Sentence encounters in a story [18]

However, the viewer suffered from several limitations. It displayed a list of records as a standard HTML table, which was not necessarily human-understandable. Although tables can be useful for comparing events of the same type, they are ill-suited to conveying the heterogeneous set of events that transpired during a given interaction, or the context in which they occurred. Navigation was restricted to drilling down from the top-level list of students or tutors, with no convenient way to specify a particular type of interaction to explore, and no visible indication of context. Finally, the viewer was inflexible. It was specific not just to the Reading Tutor but to one particular version of it. The user could not specify which events to select, how to summarize them, or (other than by deciding how far to drill down) which details to include.

1.2 Guidelines for logging tutorial interactions

Our experience led us to develop the following guidelines for logging tutorial interactions [19]. They support multiple purposes, such as using logged information in the Reading Tutor itself, generating reports for teachers [20], mining the data by constructing queries to answer research questions [15, 21], and browsing data as reported previously [1, 3] and developed further in this chapter.

1.2.1 ***Log tutor data directly to a database.***

Intelligent tutoring systems commonly record their interactions in the form of log files, and so did the Reading Tutor prior to 2002. Log files are easy to record, flexible in what information they can capture, (sometimes) human-understandable to read, and useful in debugging. However, they are unwieldy to aggregate across multiple sessions and computers, difficult to parse and analyze in ways not anticipated when they were designed, and generally cumbersome and error-prone to process [5]. Consequently, more and more researchers have discovered the hard way that easy, accurate, efficient mining of educational data requires converting it from log files into a database representation. Indeed, the viewer described in Section 1.1 operated on a database extracted rather painfully from Reading Tutor log files.

Logging tutorial interactions directly to a suitably designed and indexed database instead of to log files eliminates the need to parse them – previously a major time sink and error source for Project LISTEN. Starting with the 2003-2004 school year, the Reading Tutor has logged its interactions to a database, making the process of analyzing them easier, faster, more flexible, less bug-prone, and more powerful than analyzing conventional log files. This practice has enabled or facilitated nearly all the dozens of subsequent papers listed on the Publications page of Project LISTEN's website at www.cs.cmu.edu/~listen.

Moreover, logging straight to a database supports immediate efficient access. This capability is essential for real-time use of logged information – in particular, by the tutor itself. In fact the original impetus for re-architecting the Reading Tutor from a stand-alone application into a client of a shared database server was to free students from being constrained to use the Reading Tutor only on the particular machine that they had enrolled on and that had their records. The Reading Tutor now relies at runtime on student model information logged directly to a shared MySQL database server located on-site at the school or on campus at Carnegie Mellon. In contrast, although the PSLC DataShop uses a database, it generates the database by parsing tutor logs encoded in XML, whether as files processed after the fact, or as messages processed in real-time.

Logging directly to a database is not entirely risk-free. To mitigate the risk of database corruption, each server uses a standard MySQL feature to log each day's transactions to a file. In the event of database corruption more serious than MySQL's repair command can fix, we can use these files to regenerate the database from scratch.

1.2.2 ***Design databases to support aggregation across sites.***

To merge separate databases from multiple schools into a single database, a nightly script on the MySQL server at each site sends the day's file of transactions to our lab server, which simply reexecutes them on the aggregated database. To make this solution possible, each table must be able to combine records from different sites. Therefore the Reading Tutor uses student IDs unlikely to recur across different classes or schools, so as to distinguish data from different students. In practice, IDs that encode student gender, birthdate, and length of first and last names tend to suffice. Similarly, the Reading Tutor uses computer, ID, and start time to identify events, instead of numbering them from 1 independently at each site.

1.2.3 ***Log each school year's data to a different database.***

The database for each school year holds data logged by that year's version of the Reading Tutor from multiple computers at different schools. Each year's version of the Reading Tutor adds, omits, or revises tables or fields in the previous year's database schema, and therefore requires its own archival database. This practice accommodates differences between successive versions of the Reading Tutor, as new versions add or change fields or tables. It also facilitates running a query on one school year's data at a time, for example to develop and refine a hypothesis using data from one year, and then test it on data from other years.

Also, each team member has his or her own database to modify freely without fear of altering archival data. Making it easy for researchers to create new tables and views that are readily accessible to each other is key. This step enables best practices to propagate quickly, whereas version skew can become a problem if researchers keep private copies of data on their own computers.

1.2.4 ***Include computer, student ID, and start time as standard fields.***

The key insight here is that student, computer, and time typically suffice to identify a unique tutorial interaction of a given type. Together they distinguish the interaction from those of another type, computer, or student. (We include computer name in case the student ID is not unique, and also because some events, such as launching the Reading Tutor, do not involve a student ID.) There are two reasons this idea is powerful. First, these fields serve as a primary key for every table in the database, simplifying access and shortening the learning curve for working with the data. Second, nearly every tutor makes use of the concepts of students, computers, and time, so this recommendation is broadly applicable.

1.2.5 ***Log end time as well as start time.***

This additional information makes it possible to compute the duration of a non-instantaneous event, measure the hiatus between the end of one event and the start of another, and determine if one event starts before, during, or after another event – capabilities essential for the Session Browser, in particular to infer hierarchies of events based on temporal relations among them (as Section 3 will discuss). Many tutors log start times but not end times. For instance, logging the end time of an event can be problematic for a web-based tutor that doesn't know when the student leaves a page or window. Without information about event end times, one cannot use them to infer the hierarchical structure of events, and must instead rely on detailed knowledge of control flow in the specific tutor.

Logging the end time of an event in the same record as its start time is a bit burdensome to implement, because it requires the tutor to remember until the end of the event what information to log about it, in particular when it began. In fact to ensure that each event is logged even if the tutor crashes during it, the Reading Tutor logs the event at start time and updates it later to fill in the end time. Although logging start and end times to different records might seem simpler because it reduces the amount of state the tutor must remember at runtime, it merely replaces this requirement with the messier task of matching up start and (possibly missing) end times after the fact.

1.2.6 ***Name standard fields consistently within and across databases.***

Naming fields consistently in successive versions of the tutor makes them easier for the Session Browser to extract. Thus most of the tables in a Reading Tutor database have fields named machine_name, user_id, start_time, and (for non-instantaneous events) end_time. Timestamps in MySQL (at least in the version we use) are limited to 1-second resolution, so tables that require higher resolution encode the milliseconds portion of start and end times in separate integer-valued fields named sms and ems, respectively. It's fine to add new tables and fields as the tutor and its database schema evolve, but keeping the names of the old ones facilitates reuse of code to display and analyze tutor data.

1.2.7 ***Use a separate table for each type of tutorial event.***

Using a separate table for each event type (e.g. story, sentence, utterance, click) lets us include fields for features specific to that event type. Thus the story table includes a field for the story title; the sentence_encounter table has a field for the text of the sentence; the utterance table has a field for the filepath of the audio recording of the utterance; and the student_click table has a field for the word the student clicked on.

1.2.8 ***Index event tables by computer, student ID, and start time.***

Database indices enable fast retrieval even from tables of millions of events. For example, given computer name, student ID, and start time, retrieval from the 2003-2004 table of 10,765,345 word hypotheses output by the speech recognizer takes only 15 milliseconds – 3,000 times faster than the 45.860 seconds it would take to scan the table without using indices.

1.2.9 ***Include a field for the parent event start time.***

A field for an event's parent in the record for the event makes joins easier to write and faster to execute. For example, the sentence_encounter_start_time field of the record for a read word makes it easier and faster to find the sentence encounter containing the word than by querying for the sentence encounter that started before the word and ended after it.

1.2.10 ***Logging the non-occurrence of an event is tricky.***

"Subjunctive logging" means recording what could have happened but did not. A discussion at the ITS2006 Educational Data Mining Workshop defined this term as "what the system's other options were" (http://www.educationaldatamining.org/datalogged.html). However, it can also encompass actions the student could have done but did not [19]. Either way, tutor designers and data miners should be aware that recording some non-events can greatly simplify later analyses. For example, when the Reading Tutor gives help on a word, it chooses randomly among the types of help available. This set of choices depends on the word – for instance, the Reading Tutor cannot give a rhyming hint for the word *orange* because none exists, nor can it decompose a monosyllabic word into syllables. We compared the efficacy of different types of help based on how often the student interrupted to request a different type of help [22], or read the word fluently at the next opportunity [23]. However, a "level playing field" comparison requires comparing different types of help *on the same set of words*. Otherwise the comparison can be skewed unfairly. For example, rhyming words tend to be monosyllabic and hence easier on average than multisyllabic words. Unless carefully taken into account, this difference in scope of applicability may masquerade as a difference in efficacy that spuriously favors rhyming hints over syllabification. Unfortunately the Reading Tutor logged only the type of help it actually gave on a word, not the alternatives it could have chosen. Consequently, we had to reconstruct them based on the lexical properties of the word. Our level playing field comparisons would have been both easier and more trustworthy if each time the Reading Tutor gave help on a word by choosing randomly among the types of help available, it had logged all of them, not just the type it chose.

Just as it can be useful to log actions the tutor didn't do, it can be useful to log actions the student didn't do. For example, the Reading Tutor logs each word it heard the student read. But it is also important to know which words of text the student skipped. However, the precise time at which the student did *not* read a skipped word is undefined, so the Reading Tutor logs its start and end time as null. Fortunately, the Reading Tutor also logs the skipped word's sentence_encounter_start_time, and the position in the text sentence where it should have been read. These two pieces of information about the skipped word are non-null, enabling analysis queries and the Session Browser to retrieve words as ordered in the sentence, whether or not the student read them.

1.3 Requirements for browsing tutorial interactions

To address the limitations of the event viewer described in Section 1.1, we developed the Session Browser described in this chapter. The Session Browser is a Java™ program that queries databases on a MySQL server [24], both running on ordinary personal computers. To avoid the ambiguity of the word "user," we will instead use "researcher" to refer to a Session Browser user, and "student" to refer to a Reading Tutor user.

How should researchers explore logged interactions with an intelligent tutor? Our previous experience suggested the following requirements for the content to display (1-2), the interface to display it (2-3), and the architecture to implement the Session Browser (4):

1. Specify a phenomenon to explore.
2. Display selected events with the context in which they occurred, in dynamically adjustable detail.

3. Summarize interactions in human-understandable form.
4. Adapt easily to new tutor versions, tasks, and researchers.

The rest of this chapter is organized as follows. Sections 2-5 respectively explain how the Session Browser satisfies each requirement listed above. To illustrate its various features, we use the event depicted in Figure 2, as well as real examples based on children's tutorial interactions with Project LISTEN's Reading Tutor and our own research interactions with the Session Browser. Section 6 summarizes contributions and limitations, and evaluates the Session Browser against several criteria.

2 Specify a phenomenon to explore.

First, how can an educational data miner specify what to explore? Sections 2.1, 2.2, and 2.3 describe three different ways a researcher can indicate which tutor events to examine in the Session Browser.

2.1 Specify events by when they occurred

Place near here:

Figure 3: Event form

To explore what the Reading Tutor and student were doing on a given computer at a given point in time, the researcher can fill in the form shown in Figure 3 with the name of the computer, the ID of the student, and the date and time at that point, and click the *Process the form* button. The Session Browser then retrieves all events logged by the Reading Tutor which occurred on that computer, involved that student, and included that point in time. It performs this retrieval by querying the table for each event type for events that started before that point and ended after it. Using this feature requires knowing the precise time when the event of interest occurred. We knew the computer and user_id that generated the screenshot in Figure 2, but not exactly when. What to do?

2.2 Specify events by a database query

We need a well-defined, expressively powerful language to specify phenomena of interest and a computationally efficient search mechanism to find instances of them. Rather than invent a new language for this purpose, we exploit the expressiveness of an existing database query language [24]) and the efficiency of database retrieval. Typing in a query and clicking the *Run* button causes the Session Browser to execute the query on the database server and display the table of results it returns. The Session Browser maintains a history of these queries to make them easy to review and reuse.

Place near here:

Figure 4: Event query

To specify the event shown in the screenshot, we used the fact that it showed the Reading Tutor sounding out the word *teach.* Based on this fact, we formulated the query shown in Figure 4.

Place near here:

Figure 5: Table of events returned by query

As Figure 5 shows, the Session Browser displays the query results in a table with a numbered row for each record and a column for each field of “help_given” in the database, including the name of the computer that recorded it, the time at which the tutor decided to help, the times at which the help started and ended, the word helped, the type of help, what the tutor said or showed, the identity of the student, and so on.

To translate the result of a query into a set of tutorial events, the Session Browser scans the labels returned as part of the query, and finds the columns for student, computer, start time, and end time. As recommended in Guideline 1.2.6 above, the Session Browser assumes standard names for these columns, e.g. “user_id” for student, “machine_name” for computer, and “start_time” for start time. If necessary the researcher can apply this naming convention after the fact, e.g., by inserting “as start_time” in the query in order to relabel a column named differently.

An extensively used tutor collects too much data to inspect all of it by hand, so the first step in mining it is to select a sample. For instance, this query selects a random sample of 10 from the table of student utterances:

```
select * from utterance
order by rand()
limit 10;
```

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. For example, random examples quickly revealed Session Browser bugs not manifest when using the standard test examples used previously.

Our queries typically focus on a particular phenomenon of interest, such as the set of questions that students took longest to answer, or steps where they got 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.

For example, one such phenomenon was a particular student behavior – namely, clicking *Back* out of stories. The Reading Tutor has *Go* and *Back* buttons to navigate forward or backward in a story. In the 2004-2005 version, the *Go* button advanced to the next sentence, and the *Back* button returned to the preceding sentence. We had previously observed that students sometimes backed out of a story by clicking *Back* repeatedly even after they had invested considerable time in the story. We were interested in understanding what might precipitate this presumably undesirable behavior. This query finds a random sample of 10 stories that students backed out of after more than 60 seconds:

```
select * from story_encounter
where Exit_through = 'user_goes_back'
and unix_timestamp(end_time) - unix_timestamp(start_time) > 60
order by rand()
limit 10;
```

2.3 *Specify events by their similarity to another event*

A third way to specify events is by their similarity to an event already displayed in the event tree. Right-clicking on the event and selecting *Show command for similar* or *Show and run similar* generates a query to retrieve them, and spawns a new Session Browser window to run the query and display its results, as Figure 6 illustrates. Displaying the query lets the researcher edit it if desired.

Place near here:

Figure 6: Specifying events by similarity to another event:
Right-clicking on the highlighted help_given event in the top window and selecting "Show command for similar" brings up a new instance of the Session Browser with a query that looks for other help events with the same values as the user-selected fields of the original event, in this case machine_name, user_id, and the word *teach*:

```
select * from help_given
where machine_name= 'LISTEN07-213'
and user_id= 'mJC4-7-2002-01-01'
and word= 'teach';
```

Running this query returns the list of matching events, one of which is highlighted in the bottom window and consisted of giving the hint *Rhymes with peach.*

Here "similar events" means events that have the same type as the original event, and share the same values for the subset of fields checked off in the displayed record. For example, consider a help_requested event where the Reading Tutor sounded out a word the student clicked on. If similarity is defined as having the same user_id and word, the Session Browser can retrieve the student's history of help on that word. By selecting different subsets of fields, the researcher can direct the Session Browser to find events when the same student clicked on any word, or the Reading Tutor sounded out a word, or anyone clicked on the same word. In Figure 6, the researcher has specified "similar" as the same student on the same computer getting any type of help on the word *teach.* For example, in the highlighted event, the Reading Tutor gave the hint *rhymes with peach*.

3 Display selected events with the context in which they occurred, in adjustable detail.

As Figure 5 illustrated, the Session Browser lists field names and values for each event as a numbered row in the results table. To select one or more events from this table to focus on, the researcher uses standard click and drag mouse gestures to highlight them.

Place near here:

Figure 7: Attribute-value list for an event record

As Figure 7 shows, the Session Browser also displays all fields of the current focal event in a more spacious vertical format. However, both displays (the results table and the attribute-value list of field values) support only limited understanding of the event, because they lack context. What is the context of a set of events? Our answer is: "its chain of ancestors" – that is, the higher-level events during which they started. For example, the ancestors of a help_request event include the utterance, sentence, story, and session in which it occurred.

How can we discern this hierarchical structure of student-tutor interaction? The initial implementation of the Session Browser computed this hierarchy using its hardwired schema of the Reading Tutor database to determine which events were part of which others. This knowledge was specific to the particular tutor, increasing the work to develop the original Session Browser and reducing its generality. Moreover, such knowledge may even depend on the particular version of the tutor, making it harder to maintain the Session Browser across multiple tutor versions. At first we pondered how to declare this knowledge in a form we could input to the Session Browser. But then our programmer (Andrew Cuneo) had an elegant insight: we could eliminate the need for such declarations simply by exploiting the natural hierarchical structure of nested time intervals, which Section 3.1 now defines.

3.1 Temporal relations among events

As Figure 8 shows, we define various relations between a student or computer's events based on their time intervals.

Place near here:

Figure 8: Time intervals with various temporal relations to a focal event

---

3.1.1 The ancestors of a descendant constitute its context

- A *descendant* starts during its *ancestor* but does not include it. That is, the descendant must start after its ancestor starts, and/or end before it ends. The ancestor is part of the context in which the descendant started, even if the descendant continued after the ancestor ended.
- The *context* of one or more events is the set of all their ancestors.

3.1.2 Parents, children, and equals

- A *child* is a minimal descendant of a *parent*, and a parent is a minimal ancestor of its children, i.e. the children's ancestors do not overlap with their parent's descendants.
- If an instantaneous event occurs at the point where one ancestor ends and another begins, we define the parent as the earlier ancestor.
- *Equals* span the same time interval. They therefore share the same parents and children. As Section 4.4 will discuss, we represent annotations as pseudo-events that are equals of the events they describe.

3.1.3 Siblings

- A *younger sibling* has the same parent(s) as its *older sibling(s)* but starts later.
- An *eldest child* has no older siblings.

3.1.4 Duration and hiatus

- The *duration* of a non-instantaneous event is the difference between its start and end times.
- The *hiatus* between a parent and an eldest child is the difference between their start times.
- The hiatus between successive siblings is the difference between the older sibling's end time and the younger sibling's start time.

3.1.5 Overlapping events

What if two events overlap in time, but neither one contains the other? Section 0 above defines whichever event starts earlier as the ancestor of the other event. This relation is not transitive: just because event 2 starts during event 1 and event 3 starts during event 2 does not guarantee that event 3 starts during event 1. In order to make context trees intuitive, we first explored alternative ways to treat overlapping events.

One alternative is to treat overlapping events as siblings, i.e. require that an ancestor contain the entire descendant, not just its beginning. This approach makes the ancestor relation transitive. But then events that occur during the overlap are descendants of both siblings, and therefore appear in both their event trees when expanded, which is visually confusing. (Similar replication occurs with equal events, which have identical sets of descendants; to avoid the confusion, we are simply careful when browsing an event context to expand only one of the equals.)

To avoid such awkward replication, we decided to sacrifice the transitivity of the ancestor relation and define one overlapping event as the ancestor of the other. How to choose? Based on the intuition that the ancestor should be a larger-grain event, we at first chose whichever event was longer in duration. However, if the longer event starts later, the hiatus from the start of the ancestor to the start of the descendant is negative, which we found made it harder to understand the order of events. We therefore adopted the ancestor criterion defined in Section 0 above. Fortunately, the overlapping-event case is relatively rare in our data, thanks to the predominantly hierarchical structure of the Reading Tutor's interactions with students.

3.2 Displaying the event tree

How can we generate a dynamic, adjustable-detail view of hierarchical structure in a human-understandable, easily controllable form? Given a focal event or set of events, the Session Browser at first displays only its context, i.e., its ancestors. Thus as Figure 9 illustrated, the context of the help event depicted in the screenshot in Figure 2 includes only the session, story, and sentence during which it occurred, not any of their other descendants – except as we now explain.

Place near here:

Figure 9: Event tree shows context of the highlighted event

When the Session Browser analyzes the table of query results, it looks not only for columns named "start_time" or "end_time," but also for columns whose names end with "start_time" or "end_time." We assume that these times identify related events, so we add them to the set of selected events whose context the Session Browser will display. In this particular case, the columns come from a help_given event, which specifies a sentence_encounter_start_time but not a sentence_encounter_end_time. Therefore the Session Browser includes in the set of selected events any instantaneous event(s) with sentence_encounter_start_time as start (and end) time – namely the audio_output event "Tutor said _lets_go_on," which is represented without an end_time and is therefore instantaneous so for as the Session Browser is concerned. Its start time coincides with the end_time of the previous sentence encounter "Let me tell you why," which is therefore its parent (by the second rule in Section 3.1.2 above) and shown as such in the event tree.

3.2.1 Computing the event tree

To compute the context of a focal event, the Session Browser retrieves its ancestors from the tables for the various event types. That is, it queries each such table for events that contain the start time of the focal event. We adapted a standard expandable tree widget to display event trees. Bullets mark terminal nodes. Given a set of events to display, the Session Browser sorts it by the ancestor relation, then by start time, so that siblings are ordered from eldest to youngest. It maps each event to a node in the tree widget and links it to the nodes for its parent(s), equal(s), and child(ren).

3.2.2 Expanding the event tree

The context tree initially appears with the ancestors of the focal event(s) ancestors only partially expanded, with their other descendants omitted for brevity. The folder icon marks an event as not yet fully expanded. Clicking on the + or – next to a non-terminal node expands or collapses it, respectively.

Collapsing and re-expanding a partially expanded event reveals the rest of its children. For instance, expanding the node for the "Let me tell you why" sentence encounter in Figure 9 reveals the utterances that occurred during it, as Figure 10 illustrates. The ability to selectively expand or collapse nodes in the event tree lets the researcher drill down to events and details of interest without drowning in irrelevant details.

Place near here:

Figure 10: Hierarchical context and partially expanded details of (another) highlighted event

When the researcher first expands the current focal event by clicking on the + next to it, the Session Browser computes its children on the fly. This process is similar to computing the context of a focal event, and in fact uses the same expansion procedure, but in the opposite direction – from parent to children instead of vice versa. To retrieve the descendants of an event, the Session Browser queries the table for each event type for events that start during it.

4 Summarize events in human-understandable form.

We have already described the event trees we use to display the hierarchical structure of tutorial interaction. But how do we summarize individual events?

4.1 Temporal information

Temporal properties are common to all events, so we display them in the same way for all event types. An event's absolute start and end times seldom matter except for time of day or time of year effects. Therefore we generally display them only in the event's attribute-value list.

In contrast, the duration of an event is a simple but informative universal measure. For example, a long story encounter suggests persistence. A long sentence encounter suggests difficulty in reading it. The hiatus between two events is sometimes as informative as the events themselves because it reflects effort, hesitation, confusion, inactivity, or off-task behavior.

Precise times seldom matter for a duration or hiatus. Therefore for readability and brevity, we round them to the largest non-zero units (days, hours, minutes, seconds, or milliseconds).

Place near here:

Figure 11: What occurred before backing out of a story?

Figure 11 summarizes a real story encounter that ended with the student backing out of the story by repeatedly clicking *Back*. The fact that most of the sentence encounters just beforehand lasted 14-39 seconds reflects very slow reading, averaging a few seconds per word. The fact that the subsequent hiatuses from when the Reading Tutor displayed a sentence to when the student clicked *Back* were less than 100 milliseconds long indicates that the student was clicking repeatedly as fast as possible.

4.2 Event summaries

The complete attribute-value list for an event occupies considerable screen space, as Figure 7 illustrated. Therefore the Session Browser displays this information only for the event currently selected as the focus.

In contrast, it displays all the one-line summaries for an event tree at once. What information should such summaries include? How should it be displayed? How should it be computed? The answers depend on the type of event. We observed that although the Reading Tutor's database schema has evolved over time, the meaning of table names is nevertheless consistent across successive versions and between databases created by different members of Project LISTEN. Therefore we wrote one function for each table to translate a record from that table into a one-line string that includes whatever we think is most informative. Ideally these functions are simple enough for researchers to modify to suit their own preferences. The default string for a table without such a function is just the name of the table, e.g., "Session" or "Story_encounter." Most functions just display one or more fields of the record for the event. For example, the function for a session just shows its start time. However, some summary functions incorporate information from other tables. For example, the function for a story encounter retrieves its title from a separate table containing information about each story. The summary for an utterance retrieves aligned_word events during the utterance to show which words of the sentence were heard, as in "Then it pulls the -- -- -- --."

4.3 Audio recordings and transcription

Special-purpose code identifies events that have audio recordings logged directly to the database or to a separate .wav file, or displays "Audio not available" if the audio file has not yet been archived to our data repository. An event with audio available is marked AUDIO, as Figure 10 showed. Clicking on the node plays the audio.

Place near here:

Figure 12: Audio transcription window

Clicking the "Result Wave" tab displays the recorded audio's waveform and, if available, its transcript, as Figure 12 shows. The researcher can zoom in and out of the wave form, click and drag to select a portion of it, play it back, and add or edit its transcript, which is stored in a separate database table. Once transcribed, an event can include its transcript in its summary, as the focal event in Figure 13 illustrates.

Place near here:

Figure 13: Annotations on an event

4.4 Annotations

Right-clicking on an event and selecting Annotations brings up a table listing its current annotations, as in Figure 13. Clicking on an annotation lets the user edit or hide its value. Clicking on the blank bottom row of the *Annotate a Record* table lets the user add a new attribute.

The annotation feature is relatively recent (August 2008). So far we have used it primarily for two published analyses. One analysis examined students' free-form answers to comprehension questions [25]. The annotations specified how an expert teacher would respond, what features of the answer called for this response, and what purpose it was intended to achieve. The purpose of the analysis was to identify useful features of students' free-form answers to try to detect automatically. In the other analysis [26], two human judges hand-rated children's recorded oral reading utterances on a 4-dimensional fluency rubric, with one attribute for each dimension. The purpose of this analysis was to provide a human "gold standard" against which to evaluate automated measurements of oral reading fluency. The reason for having a second judge was to compute inter-rater reliability, so it was important for the two judges' ratings to be independent. We used the ability to hide attributes to conceal the first judge's ratings, and added a parallel set of 4 attributes for the second judge to rate the same utterances.

We implement annotations by representing them as pseudo-events in order to use the same code that retrieves regular events, integrates them into event trees, and displays them. Therefore the Session Browser stores the attribute, value, author, and date of an annotation as a record in a global table of annotations, along with the annotated event's computer, student, and time interval. An event and its annotations are equals as defined in Section 3.1.2 above, so they appear at the same depth in the event tree.

Prior to implementing the annotation mechanism, we wrote special-purpose queries to collect the information we thought the annotator would need, and exported the resulting table into an Excel spreadsheet for the annotator to fill out. For example, the Reading Tutor recorded students' free-form answers to comprehension questions [25]. We sent our expert reading teacher the transcribed answers in a spreadsheet with columns to fill in to describe how she would respond to each answer, under what circumstances, and for what purpose. We included columns for the context we thought she would need, such as which prompt the student was answering.

The annotation mechanism offers a number of advantages over the spreadsheet approach. First, it eliminates the need to write special-purpose queries (typically complex joins) to assemble the context needed by the annotator, because the Session Browser already displays event context. Second, it avoids the need to anticipate which context the annotator will require, because the annotator can use the Session Browser to explore additional context if need be. Third, storing annotations in the database instead of in a spreadsheet supports the ongoing arrival and transcription of new data. Sending spreadsheets back and forth is a cumbersome batch solution. In contrast, storing annotations directly in the database makes it easier to keep up with new data. A simple query enables the annotator to enumerate events not yet annotated.

5 Adapt easily to new tutor versions, tasks, and researchers.

Like the viewer that preceded it, the initial implementation of the Session Browser was specific to a particular version of the Reading Tutor, a particular database structure, and particular paths through and views of the data. We now describe changes we have implemented to free the Session Browser from these dependencies by making it more flexible so that it can accommodate different versions of the Reading Tutor, different data exploration tasks, and different researchers with different needs.

5.1 Input meta-data to describe database structure

How can the Session Browser 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.

The Session Browser was originally implemented with references to specific event tables, but has evolved to be more generic. When the researcher selects a database, the Session Browser queries the MySQL server for its meta-data, namely the list of tables in the database, the fields in each table, and their names and data types. Similarly, when the Session Browser queries the MySQL server for a list of events, it returns the same sort of meta-data along with the results of the query. By following the data logging guidelines in Section 1.2 above, we exploit the assumption that the names and meanings of fields are mostly consistent across database tables and over time. Thus the code assumes particular field names for student, machine, and start and end times, but overrides this convention for exceptions. For example, the normal method to extract the start time of an event looks for a field named Start_time, but is overridden for particular tables that happen to call it something else, such as Time for instantaneous types of events, or Launch_time for the table that logs each launch of the Reading Tutor.

As Section 3.2 explained, the method to compute the context of a selected target event is: First, extract its student, computer, and start time. Then query <u>every</u> table of the database for events that involve the same student and computer and contain the start of the target event. Finally, sort the retrieved records according to the ancestor relation, and display them accordingly by inserting them in the appropriate positions in the 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).

Rather than query every table to find ancestors and descendants of a given event, 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 Session Browser typically computes the context of an event with little or no delay.

5.2 Which events to include

Besides drilling down in the event tree as described in Section 3.2.2 above, the user can specify more globally which types of events to display. As Figure 14 shows, clicking on the *Pick tables* tab displays three menus in successive columns – one for databases, one for tables, and one for fields.

Place near here:

Figure 14: Select database and tables

The databases menu shows the currently selected database(s). We have a separate database for archival data from each school year. In addition, each member of our research team has a personal database so as to protect archival data when using it to build and modify new tables, which may augment archival tables with additional derived information. Selecting a personal database in addition to an archival database causes the Session Browser to include it as well when searching for an event's ancestors or descendants.

The tables menu lists the tables in the most recently selected database. We assume a database has a different table for each type of event. The checkbox for each table in the selected database specifies whether to include that type of event in the event tree. For example, turning on the *audio_output* table tells the Session Browser to include the Reading Tutor's own logged speech output in the event tree, which is an easy way to view the tutor's side of the tutorial dialogue. The ability to select which tables to include in the event tree helps focus on the types of events relevant to the particular task at hand by excluding irrelevant details.

The checkboxes do not distinguish among events of the same type. For instance, a user might want the event tree to include the tutor's spoken tutorial assistance (e.g., "*rhymes with peach*") but not its backchannelling (e.g., "*mmm*"). User-programmable filters would allow such finer-grained distinctions, but not be as simple as using check boxes to specify which event types to include. So far the check boxes have afforded us sufficient control.

Mousing over a table in the tables menu displays its list of fields in the fields menu. We use this feature primarily as a handy reminder of what fields are in a given table. However, we can also uncheck the box next to a field to omit that field when displaying the attribute-value list representation for an event record. For example, the audio_recording table includes a field for the audio input logged by the tutor. The value of this field can occupy enough memory to cause a noticeable lag in retrieving it from the database server. Unchecking the field to omit it from the display avoids this lag. In fact unlike other fields, the default setting for this particular field is to leave it unchecked.

5.3 Make event summaries customizable by making them queries

We initially implemented the summary for each type of event in Java. However, this approach made it difficult to modify event summaries on the fly. We therefore reimplemented event summaries as queries that the researcher can edit or replace by right-clicking the event and selecting *Change summary*. This operation immediately updates the summaries for all events of that type, including those already displayed. Although the event summary queries are specific to the particular structure of the database, the mechanisms for editing and interpreting them are generic, and enable researchers to change event summaries on the fly to include the aspects relevant to the particular task at hand.

Some summary functions are very simple. For example, the Session summary query is:

```
select "Session", start_time from session
where machine_name = @this.machine_name@
and user_id = @this.user_id@
and start_time = @this.start_time@
and sms = @this.sms@;
```

We added the construct @this.*field*@ to refer to a field of the event being summarized. The Session Browser replaces this construct with regular MySQL before executing the query. Combined with the generic mechanism for event duration and hiatus, this query produces summary lines such as “34 second(s) long: Session 2008-10-01 14:31:46.”

Other queries are more complex. For example, the summary line for an utterance shows text words the speech recognizer accepted in UPPER CASE, substitutions in lower case, and omitted words as --. (The omission of a word constitutes the non-occurrence of an event as discussed in Section 1.2.10.) The query to summarize an utterance is:

```
select if (center_context=1, upper(target_word),
          if (aligned_word <> '<UND>', lower(aligned_word), '—'))
from aligned_word
where Machine_name = @this.machine_name@
and utterance_start_time = @this.start_time@
and utterance_sms = @this.sms@
and user_id = @this.user_id@
order by target_word_number;
```

For example, a summary of one utterance for the sentence "Don't ya' forget to wear green on St Patrick's Day!" is:

no audio: 4 second(s) earlier, 4 second(s) long: DON'T YA start_forget(2f_axr) TO WEAR -- -- -- -- -- --

Here "start_forget(2f_axr)" is a phonetic truncation recognized instead of "forget."

5.4 Loadable configurations

The selections of which events to explore, which tables to include, and how to summarize them constitute a substantial amount of program state. The Session Browser lets the researcher name, store, and load such configurations, including which database and tables are currently selected, the summary function for each event type, the current query, and the number of the most recently selected row in the results table. When it loads the configuration, it regenerates the context of this event. The configuration does not specify which nodes to expand, but provides enough state to regenerate the event tree quickly.

At first we stored each configuration as a file on the client machine where it was generated. However, we modified the Session Browser to store configurations in the same database where it stores annotations, after finding multiple reasons to share them, as we now discuss.

5.4.1 Resume exploration

A configuration provides enough information to resume exploration at the point where it was suspended. This capability is useful for explorations too extensive to complete in a single session. Storing the configuration on the database server instead of on the Session Browser client lets researchers access it from different client machines, which makes it easier for them to share.

5.4.2 Replicate bugs

The ability to share configurations quickly proved useful in reporting and replicating bugs. Upon encountering buggy Session Browser behavior, the researcher can simply save the current configuration, send its name to the developer, and continue working. Loading the named configuration recreates a state useful (and often sufficient) for replicating the bug.

5.4.3 Support annotation by non-power-users

The ability to encapsulate and name a complex configuration makes it possible to use the Session Browser without knowing how to write queries or set up configurations. This capability enables less technically skilled personnel to use the Session Browser for task assignments such as transcription and annotation. The researcher creates and names a configuration that specifies the appropriate database(s) to query, the query to retrieve the events to transcribe or annotate, and the subset of tables to include in event trees. The transcriber or annotator simply loads the named configuration and performs the assigned transcription or annotation as described respectively in Sections 4.3 and 4.4, using the Session Browser's graphical user interface to select and browse events returned by the query. The researcher can design the query to keep track of completed vs. remaining work by restricting it to return only events still to be processed.

6 Conclusion

We conclude by summarizing research contributions, acknowledging some limitations, and evaluating the Session Browser.

6.1 Contributions and Limitations

This chapter describes an implemented, efficient, fairly general solution to a major emerging problem in educational data mining: how to explore vast student-tutor interaction logs. We identify several useful requirements for a Session Browser to support such exploration.

6.1.1 Specify a phenomenon to explore.

We frame this problem as specifying a set of events, and describe three ways to specify events: by when they occurred, by database query, or by similarity to another event. Other ways may also be useful, such as an easy way to point out a constellation of related events and search for similar constellations.

6.1.2 Display selected events with the context in which they occurred, in dynamically adjustable detail.

Our key conceptual contribution uses temporal relations to expose natural hierarchical structure. The success of this approach is evidence for specific guidelines for logging tutorial interactions, enumerated in Section 1.2. The key recommendations are **log directly to databases** (1.2.1) rather than to files, and **log events as time intervals** (1.2.5), not as instantaneous.

An event's chain of ancestors can provide informative context. However, we often want to know what happened just before a focal event in order to explore what might have precipitated it. At present, revealing this type of context involves expanding the event's ancestors more than is convenient, necessary, or desirable. Instead, it may be useful to display just the older sibling of each ancestor, and to expand it selectively. The Session Browser could locate all events that occurred within a specified time (e.g. one minute) before the focal event, but there might be too many of them to display intelligibly. It might be safer to display the last N events prior to the focal event, for some manageable value of N.

6.1.3 Summarize interactions in human-understandable form.

Screen area and human attention are scarce resources in displaying complex tutorial interactions, so it is important to summarize events clearly, concisely, and flexibly. We use duration and hiatus to expose temporal relations among events, user-editable summary queries to convey key event features, and manual annotations to add human observations.

6.1.4 Adapt easily to new tutor versions, tasks, and researchers.

We described several useful types of customization motivated by experience with the Session Browser: selecting which databases and tables to include, editing event summary functions, and defining reusable configurations. The greater the extent to which knowledge of tutor database structure is input as meta-data from the server rather than baked into the code by the developers, the more easily the Session Browser can adapt to new versions of the Reading Tutor, and perhaps eventually to other tutors.

6.1.5 Relate events to the distributions they come from

The Session Browser displays features of particular events rather than aggregate properties of sets of events. It may be useful to extend it to relate events in the event tree to the overall distribution of events in the database. How typical is the particular value of a field for the type of event in question? For instance, it may be useful not only to display the fact that a story encounter ended by backing out of the story, but to put it in perspective by including the percentage of story encounters that end in that way, whether overall or for that student, that story, that level, or some other subset of events. This notion could be applied throughout the

Session Browser to retrieve, select, expand, and describe events based on typicality or atypicality.

6.2 Evaluation

Relevant criteria for evaluating the Session Browser include implementation cost, efficiency, generality, usability, and utility.

6.2.1 Implementation cost

Implementation took only several person-weeks for the tutor-specific prototype and about the same for its generalized interval-based successor presented in 2005. Judging by the number of subsequent versions, it took roughly twice as much work since then to add improvements such as configurations, transcription, annotation, equal events, specification by similarity, user-editable event summarizers, and multi-selection of databases to query and events to display.

6.2.2 Efficiency

Running both the database server and the Session Browser on ordinary PCs, we routinely explore databases with data from hundreds of students, thousands of hours of tutorial interaction, and millions of words. The operations reported here often update the display with no perceptible lag, though a complex query to find a specified set of events may take several seconds or more.

6.2.3 Generality

Structural evidence of the Session Browser's generality includes its largely tutor-independent design, reflected in brevity of code (103 .java files totaling 799KB, with median size 4KB) and relative scarcity of references to specific tables or fields of the database. Both these measures would improve by deleting unused code left over from the initial tutor-specific implementation. Empirical evidence of generality includes successful use of the Session Browser with archival databases from different years' versions of the Reading Tutor as well as with derivative databases constructed by members of Project LISTEN. Other researchers have not as yet adapted the Session Browser to databases from tutors besides the Reading Tutor, in part because most tutors still log to files, not databases.

However, it may be feasible to use the Session Browser with data in PSLC's DataShop [16] logged by various cognitive tutors. These tutors typically log in XML, which a PSLC translation routine parses and imports into a database. Datashop users don't even need to know the structure of the database; they just need to output or convert log data into PSLC's XML format, and to be sure that the log details are sufficient.

The PSLC Datashop database has a logically and temporally hierarchical event structure (imputed by its XML translator), which generally obeys the semantic properties required by the Session Browser. Tutorial interaction could lack such structure if it consisted of temporally interleaved but logically unrelated events, or were based primarily on other sorts of relations, such as the anaphoric relation between antecedent and referent in natural language dialog. However, tutorial data from the PSLC Datashop appears to have hierarchical temporal structure. We are currently investigating the feasibility of applying the Session Browser to that data by modifying it and/or the PSLC database to use compatible representations, e.g., one table per event type, with fields for student, machine, and event and start end times, and conventions for naming them. If so, the Session Browser could apply to many more tutors – at least if their interactions have the largely hierarchical temporal structure that it exploits, displays, and manipulates.

6.2.4 Usability

The Session Browser is now used regularly by several members of Project LISTEN after at most a few minutes of instruction. Usability is hard to quantify. However, we can claim a ten- or hundred-fold reduction in keystrokes compared to obtaining the same information by querying the database directly. For example, clicking on events in the list of query results displays their context as a chain of ancestor events. Identifying these ancestors by querying the database directly would require querying a separate table for each ancestor. Moreover, the Session Browser's graphical user interface enables users to explore event context without knowing how to write queries, and stored configurations make it easy for them to retrieve events to transcribe or annotate.

6.2.5 Utility

The ultimate test of the Session Browser is whether it leads to useful discoveries, or at least sufficiently facilitates the process of educational data mining that researchers find it helpful and keep using it. We cannot as yet attribute a publishable scientific discovery to a eureka moment in the Session Browser, nor do we necessarily expect one, because scientific discovery tends to be a gradual, multi-step process rather than a single flash of insight. However, the Session Browser has served as a useful tool in some of our published research. Time will clarify its value for us and, if it is extended to work with Datashop-compatible tutors, for other researchers as well.

Acknowledgements

This work was supported in part by the National Science Foundation under ITR/IERI Grant No. REC-0326153 and in part by the Institute of Education Sciences, U.S. Department of Education, through Grants R305B070458, R305A080157, and R305A080628 to Carnegie Mellon University. Any opinions, findings, conclusions, or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the Institute, the U.S. Department of Education, or 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, the members of Project LISTEN who use the Session Browser, the contributors to the 2005 papers on which much of this chapter is based [1-3], Thomas Harris for exploring the feasibility of using the Session Browser with data from PSLC's DataShop, and Ryan Baker and the reviewers for helpful comments on earlier drafts.

References (Project LISTEN publications are available at www.cs.cmu.edu/~listen)

1. Mostow, J., Beck, J., Cen, H., Cuneo, A., Gouvea, E., and Heiner, C., An educational data mining tool to browse tutor-student interactions: Time will tell!, in *Proceedings of the Workshop on Educational Data Mining, National Conference on Artificial Intelligence*, Beck, J. E. AAAI Press, Pittsburgh, 2005, pp. 15-22.
2. Mostow, J., Beck, J., Cen, H., Gouvea, E., and Heiner, C., Interactive Demonstration of a Generic Tool to Browse Tutor-Student Interactions, in *Interactive Events Proceedings of the 12th International Conference on Artificial Intelligence in Education (AIED 2005)*, Amsterdam, 2005, pp. 29-32.
3. Mostow, J., Beck, J., Cuneo, A., Gouvea, E., and Heiner, C., A generic tool to browse tutor-student interactions: Time will tell!, in *Proceedings of the 12th International Conference on Artificial Intelligence in Education (AIED 2005)*, Amsterdam, 2005, pp. 884-886.
4. Beck, J. E., Proceedings of the ITS2004 Workshop on Analyzing Student-Tutor Interaction Logs to Improve Educational Outcomes, Maceio, Brazil, 2004.
5. Mostow, J. and Aist, G., Evaluating tutors that listen: An overview of Project LISTEN, in *Smart Machines in Education*, Forbus, K. and Feltovich, P. MIT/AAAI Press, Menlo Park, CA, 2001, pp. 169-234.

6. Mostow, J. and Beck, J., When the Rubber Meets the Road: Lessons from the In-School Adventures of an Automated Reading Tutor that Listens, in *Scale-Up in Education*, Schneider, B. and McDonald, S.-K. Rowman & Littlefield Publishers, Lanham, MD, 2007, pp. 183-200.

7. Mostow, J., Aist, G., Burkhead, P., Corbett, A., Cuneo, A., Eitelman, S., Huang, C., Junker, B., Platz, C., Sklar, M. B., and Tobin, B., A controlled evaluation of computer- versus human-assisted oral reading, in *Artificial Intelligence in Education: AI-ED in the Wired and Wireless Future*, Moore, J. D., Redfield, C. L., and Johnson, W. L. Amsterdam: IOS Press, San Antonio, Texas, 2001, pp. 586-588.

8. Mostow, J., Aist, G., Huang, C., Junker, B., Kennedy, R., Lan, H., Latimer, D., O'Connor, R., Tassone, R., Tobin, B., and Wierman, A., 4-Month evaluation of a learner-controlled Reading Tutor that listens, in *The Path of Speech Technologies in Computer Assisted Language Learning: From Research Toward Practice*, Holland, V. M. and Fisher, F. P. Routledge, New York, 2008, pp. 201-219.

9. Poulsen, R., Wiemer-Hastings, P., and Allbritton, D., Tutoring Bilingual Students with an Automated Reading Tutor That Listens, *Journal of Educational Computing Research* 36 (2), 191-221, 2007.

10. Mostow, J., Aist, G., Bey, J., Burkhead, P., Cuneo, A., Junker, B., Rossbach, S., Tobin, B., Valeri, J., and Wilson, S., Independent practice versus computer-guided oral reading: Equal-time comparison of sustained silent reading to an automated reading tutor that listens, in *Ninth Annual Meeting of the Society for the Scientific Study of Reading*, Williams, J., Chicago, Illinois, 2002.

11. Aist, G., Towards automatic glossarization: Automatically constructing and

administering vocabulary assistance factoids and multiple-choice assessment, *International Journal of Artificial Intelligence in Education* 12, 212-231, 2001.

12. Mostow, J., Beck, J., Bey, J., Cuneo, A., Sison, J., Tobin, B., and Valeri, J., Using automated questions to assess reading comprehension, vocabulary, and effects of tutorial interventions, *Technology, Instruction, Cognition and Learning* 2, 97-134, 2004.
13. Mostow, J., Beck, J. E., and Heiner, C., Which Help Helps? Effects of Various Types of Help on Word Learning in an Automated Reading Tutor that Listens, in *Eleventh Annual Meeting of the Society for the Scientific Study of Reading*, Reitsma, P., Amsterdam, The Netherlands, 2004.
14. Corbett, A. and Anderson, J., Knowledge tracing: Modeling the acquisition of procedural knowledge, *User modeling and user-adapted interaction* 4, 253-278, 1995.
15. Beck, J. E. and Mostow, J., How who should practice: Using learning decomposition to evaluate the efficacy of different types of practice for different types of students, in *9th International Conference on Intelligent Tutoring Systems*, Montreal, 2008, pp. 353-362. Nominated for Best Paper.
16. Koedinger, K. R., Baker, R. S. J. d., Cunningham, K., Skogsholm, A., Leber, B., and Stamper, J., A Data Repository for the EDM community: The PSLC DataShop, in *Handbook of Educational Data Mining*, Romero, C., Ventura, S., Pechenizkiy, M., and Baker, R. S. J. d. CRC Press, Boca Raton, FL, this volume.
17. Shavelson, R. J. and Towne, L., Scientific Research in Education, National Academy Press, National Research Council, Washington, D.C., 2002.
18. Mostow, J., Beck, J., Chalasani, R., Cuneo, A., and Jia, P., Viewing and analyzing multimodal human-computer tutorial dialogue: a database approach, in *Proceedings of*

the Fourth IEEE International Conference on Multimodal Interfaces (ICMI 2002) IEEE, Pittsburgh, PA, 2002, pp. 129-134. First presented June 4, 2002, at the ITS 2002 Workshop on Empirical Methods for Tutorial Dialogue Systems, San Sebastian, Spain.

19. Mostow, J. and Beck, J. E., Why, What, and How to Log? Lessons from LISTEN, in *Proceedings of the Second International Conference on Educational Data Mining*, Córdoba, Spain, 2009, pp. 269-278.
20. Alpern, M., Minardo, K., O'Toole, M., Quinn, A., and Ritzie, S., Unpublished Group Project for Masters' Lab in Human-Computer Interaction, 2001.
21. Beck, J. E., Chang, K.-m., Mostow, J., and Corbett, A., Does help help? Introducing the Bayesian Evaluation and Assessment methodology, in *9th International Conference on Intelligent Tutoring Systems*, Montreal, 2008, pp. 383-394. ITS2008 Best Paper Award.
22. Heiner, C., Beck, J. E., and Mostow, J., When do students interrupt help? Effects of time, help type, and individual differences, in *Proceedings of the 12th International Conference on Artificial Intelligence in Education (AIED 2005)*, Looi, C.-K., McCalla, G., Bredeweg, B., and Breuker, J. IOS Press, Amsterdam, 2005, pp. 819-826.
23. Heiner, C., Beck, J. E., and Mostow, J., Improving the help selection policy in a Reading Tutor that listens, in *Proceedings of the InSTIL/ICALL Symposium on NLP and Speech Technologies in Advanced Language Learning Systems*, Venice, Italy, 2004, pp. 195-198.
24. MySQL, Online MySQL Documentation at http://dev.mysql.com/doc/mysql, 2004.
25. Zhang, X., Mostow, J., Duke, N. K., Trotochaud, C., Valeri, J., and Corbett, A., Mining Free-form Spoken Responses to Tutor Prompts, in *Proceedings of the First International Conference on Educational Data Mining*, Baker, R. S. J. d., Barnes, T., and Beck, J. E., Montreal, 2008, pp. 234-241.

26. Mostow, J. and Duong, M., Automated Assessment of Oral Reading Prosody, in *Proceedings of the 14th International Conference on Artificial Intelligence in Education (AIED2009)*, Dimitrova, V., Mizoguchi, R., Boulay, B. d., and Graesser, A. IOS Press, Brighton, UK, 2009, pp. 189-196.

Tables

Table 1: Activities in a session [18]

The table contains a row for each activity in the session, and a column for each field displayed: the start time of the activity; the number of sentences read; the number of sentences in the story (including word previews and reviews before and after the story proper); the story title; and how the story encounter ended. Information about time spent on the story, reader level, story level, and who picked the story has been omitted here to save space.

Start Time	NumSent Encount	NumSentences	Title	Exit Through
04-05-2001 12:24:25	6	40	How to Make Cookies by Emily Mostow.	end_of_activity
04-05-2001 12:28:14	14	56	One, two,	end_of_activity
04-05-2001 12:31:34	5	112	Pretty Mouse by Maud Keary	select_response

Table 2: Sentence encounters in a story [18]

This table has a row for each sentence encounter in the activity, including individual word previews and reviews, and a column for each field shown, namely the sentence encounter's start time and duration, the number of tutor and student actions during it, the number of student utterances recorded, and the text of the word or sentence.

Start Time	Duration	Num Actions	Num Utterances	SentenceStr
04-05-2001 12:24:25	00:00:01	3	0	OVEN
04-05-2001 12:24:27	00:00:01	3	0	BATTER
04-05-2001 12:24:28	00:00:44	47	4	First get the batter
04-05-2001 12:25:12	00:00:24	20	4	Next put all the ingredients in
04-05-2001 12:25:37	00:00:33	3	2	Then put it in the oven
04-05-2001 12:26:11	00:00:40	3	3	Last eat them

List of Figures

NOTE to publisher: Figures are in separate files. Filename = author + caption, e.g. "Mostow Figure 1 Picking a story in Project LISTEN's Reading Tutor."