Viewing and Analyzing Multimodal Human-computer Tutorial Dialogue: A Database Approach

Jack Mostow, Joseph Beck, Raghu Chalasani, Andrew Cuneo, and Peng Jia

Project LISTEN, Carnegie Mellon University, Pittsburgh, USA
email: {mostow, raghu, acuneo, pengj}@cs.cmu.edu; joseph.beck@cmu.edu

Abstract

It is easier to record logs of multimodal human-computer tutorial dialogue than to make sense of them. In the 2000-2001 school year, we logged the interactions of approximately 400 students who used Project LISTEN's Reading Tutor and who read aloud over 2.4 million words. This paper discusses some difficulties we encountered converting the logs into a more easily understandable database. It is faster to write SQL queries to answer research questions than to analyze complex log files each time. The database also permits us to construct a viewer to examine individual Reading Tutor-student interactions. This combination of queries and viewable data has turned out to be very powerful, and we discuss how we have combined them to answer research questions.

1. Introduction

It is easier to record logs of multimodal human-computer tutorial dialogue than to make sense of them. We discuss this problem in the context of Project LISTEN's Reading Tutor, which listens to children read, and helps them [1]. The resulting multimodal dialogue includes mouse clicks and speech as student input, and spoken and graphical assistance as tutor output. In the 2000-2001 school year, hundreds of students used the Reading Tutor daily at three elementary schools, reading over 2.4 million words. How can we analyze so much data? The Reading Tutors logged thousands of sessions, but the logs are too detailed to see the forest for the trees.

An alternate way to capture detailed tutorial interactions in human-viewable form is to videotape them, as we have done in a number of studies [2]. Video has obvious advantages, but many drawbacks. It is laborious to record at schools, invades privacy, can distort student behavior, captures only one level of externally observable detail, omits internal events and tutorial decision processes, and is tedious to analyze. To avoid these drawbacks, we describe a database approach we developed to view and analyze logs of children's interactions with Project LISTEN's Reading Tutor.

A Reading Tutor session consists of logging in followed by a series of stories, which the student and Reading Tutor take turns picking. Each story is a sequence of a few types of steps: assisted reading, writing, listening, and picking. The Reading Tutor may insert a preview activity beforehand, and/or a review afterward, also built out of the same types of steps.

The Reading Tutor's output is textual, graphical, and audio. It displays text on the computer screen, sentence by sentence, for the child to read aloud. It gives help on a word or sentence by playing human or synthesized speech, with graphical cues such as highlighting words.

The Reading Tutor inputs the student's speech, mouse clicks, and keyboard presses. It uses a less than perfectly accurate speech recognizer to produce a time-aligned transcript of each student utterance.

2. Data recorded by the Reading Tutor

The Reading Tutor logs information in various forms. Although other multimodal dialogue systems may be organized very differently, we suspect that they may encounter some of the same problems we identify, such as proliferation of log entry types, the need to identify a few key control points, and the need to identify the appropriate level(s) of detail and aggregation at which to represent and analyze tutorial interactions.

The .wav file for each utterance contains the digitized speech. Its "hypothesis" file contains the speech recognizer's time-aligned transcript, one word per line, showing its start and end time as offsets in centiseconds from the beginning of the utterance:

```
BY 227 249
11 250 278
```

Utterance files are a simple representation, both human-readable and conducive to automated analysis, especially since this representation has remained stable for years. However, they are incomplete, capturing only what students said and what the recognizer "heard," not

what else the student or Reading Tutor did. The "sentence" file shows the alignment of this hypothesis against the sentence text, using –1 to indicate that a sentence word had no hypothesis word aligned against it:

```
121     -1      -1
divided -1      -1
by      227     249
11      250     278
is      -1      -1
11      -1      -1
```

The Reading Tutor records tutorial interactions in logs, starting a new log each time it is launched. Log files are tens of megabytes in size and thousands of lines long. Each line is generated by a "logprint" function that records a sequential event number, a severity level, a timestamp, the number of digitized speech samples recorded so far, the name of the function that logged this event, and an event description whose form and content depend on the type of event logged and is often much longer than in the examples, which were chosen for their brevity. For example, this pair of lines in the log indicates that an utterance has ended and been captured:

```
16466, Notice, "Tue Apr 10 12:30:20.387
   2001", 10763200,
   "CListener::FinalizeUtterance",
   "EndUtterance"

16467, Notice, "Tue Apr 10 12:30:20.417
   2001", 10763200,
   "CCapture::WriteWaveFile(int)",
   "Wrote File:  d:\\listen\\cd\\Tue-Sep-
   19-23-44-58.093-2000\\Capture\\fAT6-6-
   1994-08-01\\dec-fAT6-6-1994-08-01-
   Apr10-01-12-30-14-902.wav"
```

Over years of development, the Reading Tutor code has accumulated over a thousand calls to logprint for various purposes, including debugging and performance tuning. These calls record not only externally observable events, but also internal decisions at various levels of control. For example, suppose the student clicks on the word "cat." The Reading Tutor first computes the types of help it can give on this word, such as speaking the word, sounding it out phoneme by phoneme, and so forth. From this set it chooses to give, say, a rhyming hint. From its table of rhymes, it randomly chooses the word "bat," and verifies that it has a recording of "bat." Then it queues a sequence of audio and graphical actions to say "rhymes with," display the word "bat" beneath the word "cat," and say "bat." The log file records this sequence of actions as separate events, but does not explicitly link them together as a single abstract event of the form "give help of type *h* on word *w* from time *t1* to *t2*." One reason for this omission is the difficulty in logging intervals rather than single events. Obviously a log message cannot be generated at time *t1* that also provides the end time. Logging the event at time *t2* would destroy the chronological ordering of the log files. Since the log files were designed to be human-readable, this ordering is an important property.

Although a detailed record can be useful for debugging, it is impractical to write scripts to parse and analyze a thousand different types of log entries. We also experimented with the Reading Tutor generating html summaries of students' interactions [3]. Although they were more human-readable than logs, their fixed level of detail did not support subsequent changes to form or content, or more powerful analysis than simple browsing.

3. A database approach

A July 2000 talk by Dr. Alex Rudnicky on his session browser for the Communicator system [4], coupled with our desire to understand how students were using the Reading Tutor at schools, inspired the vision of a log viewer to display Reading Tutor interactions at multiple levels of detail. We report on work in progress toward this vision – bugs and all.

Our approach parses utterance and log files into a database, enabling us to answer research questions by writing concise database queries instead of complex perl scripts. The log viewer uses stored queries to generate views at different levels dynamically, allowing us to modify the view form and content. We now explain how we represent, populate, query, and view the database.

3.1. Database schema for Reading Tutor data

Figure 1 summarizes our schema (simplified due to space limitations) to model data for the 2000-2001 school year from hundreds of students at three elementary schools. Database schemas are important to get right, and ours took weeks to finalize.

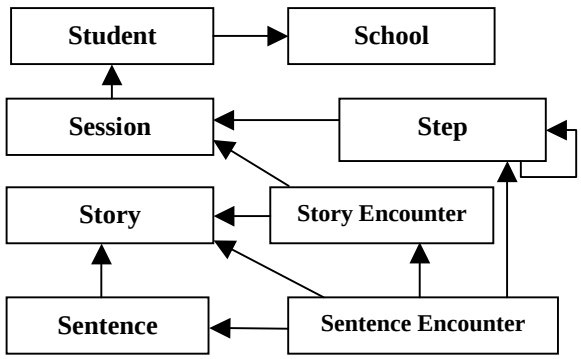

Figure 1: Abridged database schema of Reading Tutor interactions

The entities modeled in the schema range in grain size from schools down to individual word encounters. Arrows encode many-to-one relations. For example, a school had multiple students; students had multiple sessions with the Reading Tutor.

We faced a challenge in figuring out how to model the elements of a session. One choice was to model a session

as a sequence of story encounters, a story encounter as a sequence of sentence encounters, and a sentence encounter as a sequence of student utterances. This model matched a useful simplified view of a session as consisting of picking a series of stories to read, and it used appropriately different fields to describe stories, sentences, and utterances. For example, story tables had a field for the difficulty level of the story; sentence encounter tables had a field for when the student began reading this sentence; and utterance tables had a field for the name of the wav file that contained the student's speech. However, this structure did not match the reality that stories were multi-step activities, that the Reading Tutor often inserted a teaching activity before a story and/or a review afterwards, and that the Reading Tutor architecture treated each session as a uniform tree structure of such steps. Such a step tree would be easy to represent recursively in the schema, but would omit bona fide distinctions among stories, sentences, and utterances. To reconcile these two models, we decided to encode both in parallel, relating them by linking each sentence encounter to the step during which it occurred.

We used separate tables to represent mouse clicks, graphical actions, and Reading Tutor audio output because they had different fields. However, it is not very useful to view them separately, so we generated merged tables that combined them into a unified sequential view.

We did not model the Reading Tutor's decision processes, only its interactions with students. We decided to model feedback as occurring either before the student began to read the sentence, after the Reading Tutor detected a mistake, as backchanneling (active listening such as "uh huh"), or as praise.

3.2. Populating the database

To populate the database, a perl script parsed through individual log files in a single forward pass.

When the parser generated a record, it called a filer function to handle the communication with the database and create the necessary records in dependent tables, such as Sentence for a sentence encounter record. The filer returned an internal id from the database so that future related records, such as audio output within a sentence encounter, could refer to it.

For example, this line marked the start of a sentence encounter with the text "Dividing By 11":

```
15023, Notice, "Tue Apr 10 12:27:13.648
  2001", 7776000,
  "CParagraph::SetSentence", "Dividing By
  11"
```

Accordingly, the script created a Sentence Encounter record with 12:27:13.648 as its start time, "Dividing By 11" as its text, and (based on state maintained from parsing previous lines) the internal (database) ids of the launch, session, story encounter, and step. 740 lines later, the script found a "coach_goes_forward" event, which enabled it to fill in the end time of the Sentence Encounter as 12:28:35.386.

Similarly, the parser started an Utterance record when it encountered a "user_begins_turn" event, and completed the record when it encountered a pair of lines in the log showing that an utterance ended and was captured.

3.3. Multi-level log viewer implementation

Given the database, the viewer was straightforward to implement. We used MySQL Database-Server to serve the database, perl DBI packages to interface to the database, perl and CGI scripts to generate the views, and Apache Web-Server to serve the views. All of these packages are free to download. To save time, we adopted a uniform tabular style rather than craft more specialized, aesthetic views, as we would if they were intended for teachers and not just ourselves.

A link to a view is encoded as a call to the script that generates that view. For example, clicking the link http://logviewer.cmu.edu:9876/cgibin/storyList.pl?session_id=8562 calls the script storyList.pl to list the story encounters for session number 8562. The script executes a database query to retrieve them:

```
select story.story_id, title, level,
  file_path, story_encounter_id,
  start_time, end_time, type_desc,
  student_level, byte_offset,
  event_number, initiative, sms, ems
from type_description, story,
  story_encounter
where story_encounter.story_id =
    story.story_id and
  story_encounter.session_id =
    $sessionid and
  type_description.type_id =
    story_encounter.exit_through
order by start_time, sms
```

The script outputs an HTML table with a row for each record returned, embedding any links to other views.

3.4. Views: what the browser shows

In general, the log viewer generates a view as a list of records in an HTML table, with one row per value, and column headers at the top. One or more fields in each row may contain clickable links to more detailed views. To protect student anonymity, we restrict access and omit or alter names in our examples.

The highest-level view lists the (three in this database) values of School, with columns for school name, location, number of Reading Tutors, and number of students. Clicking on a school name brings up a table of Reading Tutors at that school, with columns for machine name, description, and number of launches. Clicking on a machine name brings up a chronologically ordered table

with one row for each launch of the Reading Tutor on that machine. Each row includes the date and time of the launch, which version of the Reading Tutor was launched, the number of sessions during that launch, and a link to the log file for the launch. Clicking on a session count links to the list of sessions, and so on.

Clicking on a session's story count, step count, or student name brings up a table of stories read, session steps, or student information, respectively. For example, Figure 2 lists the stories read in one session. Each row shows the story encounter's start time, number of sentences read, number of sentences in the story, title, and how the 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.

This example revealed two problems. A bug in the populating script inflated the number of sentences in the story by including previews and reviews. Second, the "select_response" value for how the last story encounter ended indicates that the log was missing expected information. As these examples illustrate, a database populated from log files not designed for that purpose can be informative, but buggy or incomplete.

Start Time	NumSent Encount	NumSe ntences	Title	Exit Throug h
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_re sponse

Figure 2: Table of activies for a session

Following the sentence encounters link in the first story listed, "How to Make Cookies by Emily Mostow," brings up Figure 3. The first two records come from preview activities that introduced the words "oven" and "batter," which the student was encountering for the first time in the Reading Tutor. The fifth record shows that the sentence "Then put it in the oven" took 33 seconds, with 2 utterances and 3 other actions, totaling 5 – few enough to list here, which is why we chose this example.

Figure 4 shows what happened during this encounter. First the Reading Tutor decided to give some preemptive assistance, though exactly what is not specified. 9 seconds later it prompted the student by saying "to get help click, on a word." The two utterances turned out to consist of the words "then put," followed by off-task speech. Finally the student clicked the *Go* button to go on to the next sentence.

Start Time	Duration	Num Actio ns	Num Utter ances	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

Figure 3: Table of sentence encounters for the story "How to Make Cookies by Emily Mostow"

Besides starting at the list of schools and browsing downward to more detailed views, we wanted to view specific entities found by queries. We therefore provided a more direct form of access by inputting the entity's database id. The same integer represents different ids in different tables, so the user must also select the type of table – of schools, Reading Tutors, launches, sessions, story encounters, sentence encounters, or utterances.

Start Time	End Time	Action	Description
04-05-2001 12:25:38	00-00-0000 00:00:00.0	Abstract Response	preemptive help
04-05-2001 12:25:47	00-00-0000 00:00:00.-1	Audio	NONE
04-05-2001 12:25:48	04-05-2001 12:26:00	Utterance	NONE
04-05-2001 12:26:01	04-05-2001 12:26:11	Utterance	NONE
04-05-2001 12:26:11	00-00-0000 00:00:00.-1	Click	user_goes_fo rward

Figure 4: Table of actions and utterances for the sentence encounter "Then put it in the oven"

4. Answering questions with the database

We are using the database both to replicate previous studies [5] and to answer new research questions [6], thanks to the (relative) ease of constructing queries and

validating their correctness. For statistical analysis we use SPSS's ability to import from the database and our SQL client (urSQL)'s ability to export results into Excel.

4.1. Constructing queries

To answer a research question, we formulate it as an SQL query. As a simple example, were students likelier to back out of a story if the Reading Tutor chose it than if they did? This query counts how often students backed out of stories, disaggregated by who chose:

```
select se.initiative, count(*)
     from student_click sc, story_encounter se,
   sentence_encounter sen
     where
       sc.start_time = se.end_time and
       sc.type_id=8 and
       sc.sentence_encounter_id =
         sen.sentence_encounter_id and
       sen.story_encounter_id =
         se.story_encounter_id
     group by se.initiative
```

The basic logic (in the `where` condition) is to find story encounters that ended when the student clicked *Back* (click type 8) out of a sentence and this click occurs at the same time a story ended. One problem could arise if a student clicked back at the same time someone else finished a story on a different computer. This case should not count as backing out of a story. To avoid this miscount, the `where` clause also specifies that the click must occur in a particular sentence in the story that ends at the same time as the click. The first line specifies what data to collect – the initiative (who chose the story) and how many items met the condition.

The query results in Table 1 (based on about 400 students) support a Yes answer to the research question: the Reading Tutor chose the story in 2457 cases where the student backed out, the student chose in 703, and who chose was not specified in 140 cases. Null values may be deliberate for introductory tutorial "stories" presented automatically to newly enrolled students, or may be caused by bugs in generating or parsing the logs. They seem inevitable in a large database, but if rare enough do not prevent analysis, as this example illustrates.

Table 1: Results of query to determine when students backed out of stories

Item	Initiative	count(*)
1	(null)	140
2	student_initiative	703
3	tutor_initiative	2457

4.2. Using the viewer to debug queries

Queries are excellent tools for generating summary results, but are less powerful for examining individual cases. Queries therefore complement the browser, whose strength is its ability to display individual cases. For example, to analyze how much time students spent waiting for the Reading Tutor to respond, we developed a query to compute the delay from the last word a student says in a sentence to the first word s/he says in the next sentence. This query is 75 lines long because it requires several steps: finding the last word of a sentence and the time it was uttered, finding the time the student uttered the first word of the next sentence, ensuring that both sentences are in the same story, etc. After debugging each step, we ran the full query. By sorting the resulting delays, we found some too long to be believable. The table of results identified the story encounters where they occurred. By using the viewer to browse these story encounters, we found the bug: between some sentence encounters were activities where students were supposed to write, which the query erroneously included as part of the delay. It is important to note that it is not necessary to create a new view for each research question. For example, the view shown in Figure 3 was used to find the flaws in the query to compute how long students were waiting. However, this view was designed before we knew that we would be conducting this analysis, and could be used to verify other queries.

An additional benefit of the log viewer is that by presenting student-computer interactions in a more understandable form, it helps people with incomplete knowledge of the project to take part in data analysis. Our project had a near perfect split between those people who understood how the Reading Tutor worked and those people who could write SQL queries. The viewer allowed those unfamiliar with the tutor to perform sanity checks on their queries (as in the case of student writing activities, mentioned above). People who were less familiar with SQL used the viewer to examine unlikely query results and to find glitches in the database. This dichotomy of project members' knowledge is not unique to Project LISTEN; finding some means to work around this gap is very helpful.

4.3. Benefits of using a database

One benefit of using the database is the ease of extracting summary information. We have had project meetings where questions were raised and immediately addressed by a quickly written query (e.g. "How many times did the Reading Tutor provide each type of help?"). Although is it slower to get detailed information about each student, rather than a summary, from the database, it is comparable to or better than the prior technique of using perl scripts. The comparison of SQL vs. perl is not quite a fair one, as the difference in ease of use has less to do with the languages than with the data each of them processes. SQL queries manipulate a structured database that we took time to set up, while the perl scripts had to work with low-level log files. To create the database, we

had to debug a set of perl scripts. This task was time intensive, but only had to be done once. Its constant cost is amortized over all of the analyses performed. So for investigating a small set of research questions a database might not be worth the cost of setting up, but for more open-ended investigations it is.

5. Conclusions

We have described a database approach to view and analyze multimodal tutorial dialogue, and how we applied it to Reading Tutor data. We now summarize its caveats, then its benefits.

It was hard to develop a good schema and useful views, especially for pre-existing logs that lack some desired information, at least in easy-to-extract form. We now make the Reading Tutor generate the database records in real time. Deriving a schema ahead of time is simpler in some ways, but could result in critical information being lost if the schema is poorly designed.

The database must be robust to tutor crashes and bugs. For example, when a crash ends a log prematurely, the end time of events in progress must be filled in. Parsing the logs exposed some Reading Tutor bugs, such as assigning the same filename to two utterances, which must not be allowed to corrupt the database.

Populating the database took weeks for our data, with 2.4 million word encounters. In the 2000-2001 school year, 33 Reading Tutors each recorded hundreds of logs, typically tens of megabytes long, with thousands of utterance files. To avoid duplicating data, a long populating process must be robust to stops and restarts.

Although the database takes long to design and construct, it pays off in queries much shorter than perl scripts, because they are expressed more declaratively. Database technology absorbs much of the complexity of searching and assembling data. When necessary, we speed up queries by adding appropriate indices, but that type of optimization is easier and less bug-prone than rewriting conventional procedures to speed them up.

Views package specific queries in an understandable form, easier to use than querying the database directly – especially for views that integrate multiple tables, and for users more fluent at clicking on links than at formulating SQL queries. It is hard to design views both concise and detailed enough to be useful. Views should summarize lower-level details in informative aggregate form, for example, durations and counts of utterances and actions in a sentence encounter. Queries make such aggregation easier, less bug-prone, and more flexible than in procedural tutor code.

We use queries both to answer statistical questions by aggregating over lots of data, and to find examples of particular phenomena, such as outlier values. We use the viewer to inspect such examples in detail, finding bugs or unexpected cases that refine the question. Finally, simply browsing our data at multiple levels often exposes interesting phenomena.

Acknowledgements

This paper is a revised and abbreviated version of [3], whose reviewers we thank for their helpful comments. We also thank other members of Project LISTEN who contributed to this work; mySQL's developers; and the students and educators at the schools where Reading Tutors recorded data.

This work was supported by the National Science Foundation under Grant Number REC-9979894. 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.

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

1. Mostow, J. and G. Aist. Evaluating tutors that listen: An overview of Project LISTEN. In K. Forbus and P. Feltovich, Editors, *Smart Machines in Education*, 169-234. MIT/AAAI Press: Menlo Park, CA, 2001.

2. Mostow, J., C. Huang, and B. Tobin. Pause the Video: Quick but quantitative expert evaluation of tutorial choices in a Reading Tutor that listens. In J.D. Moore, C.L. Redfield, and W.L. Johnson, Editors, *Artificial Intelligence in Education: AI-ED in the Wired and Wireless Future*, 343-353. Amsterdam: IOS Press: San Antonio, Texas, 2001.

3. 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.

4. Bennett, C. and A.I. Rudnicky. The Carnegie Mellon Communicator Corpus. *7th International Conference on Spoken Language Processing (ICSLP2002)* 2002. Denver, Colorado.

5. Mostow, J. and G. Aist. The Sounds of Silence: Towards Automated Evaluation of Student Learning in a Reading Tutor that Listens. *Fourteenth National Conference on Artificial Intelligence (AAAI-97)*, 355-361. 1997. Providence, RI: American Association for Artificial Intelligence.

6. Mostow, J., G. Aist, J. Beck, R. Chalasani, A. Cuneo, P. Jia, and K. Kadaru. A La Recherche du Temps Perdu, or As Time Goes By: Where does the time go in a Reading Tutor that listens? *Proceedings of the Sixth International Conference on Intelligent Tutoring Systems (ITS'2002)*, 320-329. 2002. Biarritz, France: Springer.