A data collection framework for capturing ITS data based on an agent communication standard

Olga Medvedeva, MS, MS [1], Girish Chavan, MS [1], and Rebecca S. Crowley MD, MS [1,2,3]

[1] Centers for Oncology and Pathology Informatics, University of Pittsburgh School of Medicine, Pittsburgh PA
[2] Center for Biomedical Informatics, University of Pittsburgh School of Medicine, Pittsburgh PA
[3] Intelligent Systems Program, University of Pittsburgh, Pittsburgh PA
medvedevaop@upmc.edu and crowleyrs@upmc.edu

Abstract

We describe our experience developing a flexible architecture for collecting interaction data for a medical cognitive tutoring system. The architecture encompasses (1) an agent-based communications system for passing messages between all tutor components, as well as capturing and storing them in relational format, (2) a schema for managing all system data from low-level interface events to student-tutor interaction and experimental variables, and (3) an interface for querying and retrieving this data. The system has been in use over the past year, and includes data from one large study and several smaller studies. We discuss some of the lessons we have learned over the past year as we strive to achieve a scalable and maintainable system to support educational data mining in our domain. We also argue that a standards based approach to messaging could facilitate development of shared data sets, and especially shared analytic services for the next generation of tutoring systems.

Introduction

The purpose of this manuscript is to describe our attempts to create a flexible data collection framework based on a known communication standard. What could this possibly have to do with shared data sets, meta-analysis, or educational data-mining? We argue that one of the most significant barriers to educational data mining is the lack of common resources for analysis and modeling of these data sets. Typically, the developers of each new system begin from scratch, creating the system and then all of the associated user modeling, data-mining, or other analytic resources to support the research project. This slows progress and also limits the generality of lessons learned. An important barrier to sharable resources is the absence of communication or data standards.

As noted by the workshop organizers, many intelligent educational systems are based on a common underlying theory – for example cognitive tutors (Anderson et al. 1995) or constraint-based tutors (Mitrovic et al. 2001). When the paradigm is well established – it is not too outlandish to envision separate tutoring systems sharing common services. For example, two tutoring systems in different domains and different locations could share a common resource for knowledge tracing, or for evaluating student help request behavior. What is needed to move us towards such sharable resources? One important step would be to advance a standard method for information exchange. We suggest that the Foundation for Intelligent Physical Agents (FIPA) (http://www.fipa.org) standard encompasses many of the features necessary to provide a *lingua franca* for educational systems that share a common underlying theory. We detail our use of the FIPA standard to develop a framework for data collection, and suggest how the adoption of a standard might reduce the barriers to shared data and shared resources.

Background

In the past, most attempts to support distributed ITS were based on a peer-to-peer model of communication (Brusilovsky, Ritter, Schwarz 1997, Ritter 1997). Typically this meant that (1) there was a very strict communication channel – the server for one system could not easily communicate with an analytic process for another system unless the relationship is known in advance, and (2) there was a 1:1 translation between the message representation for different systems – therefore communication among N systems required N(N-1) translators. Ultimately, the lack of a standard communication protocol greatly limited the potential for developing centralized resources (Koedinger, Suthers, Forbus 1999).

There are few existing, well-documented methods for inter-system communication and data collection among ITS (Ritter, Koedinger 1996). The Dormin communication protocol, developed at CMU, allows for message exchange between user interface and tutoring system (http://ctat.pact.cs.cmu.edu/tikiindex.php?page= DorminMessages). Dormin communication language elements include: verb, hierarchical description of receiver, message content, and message id. Dormin communication language and protocol are used in both tutoring systems and tutor authoring systems (Ritter, Blessing, Wheeler 2003). Although Dormin messages embody many of the characteristics needed in a messaging standard, they are not widely used.

In agent-based architectures, a society of independent communicating agents work together to meet a set of goals. Agent based architectures are now relatively common among tutoring systems, providing a means for collaboration in distributed ITS. Emerging agent standards may provide an opportunity for a more widely acceptable communication standard. FIPA (http://www.fipa.org) is a collection of standards for agent technology, including (1) agent management system, and (2) communication specifications. The basic message format for FIPA is shown in Figure 1. FIPA specifies the minimum required elements but permits introduction of new elements. For example, the FIPA performatives (which function much like Dormin verbs) specify a set of 22 common communication acts, such as accept-proposal, confirm, disconfirm, failure, reject-proposal, and subscribe. However, additional performatives can be added by adding an X- to the beginning of the performative. In effect, this provides a controlled vocabulary of communication acts to aid in inter-system communication. The existing set of performatives could be extended by a community to serve community-specific needs.

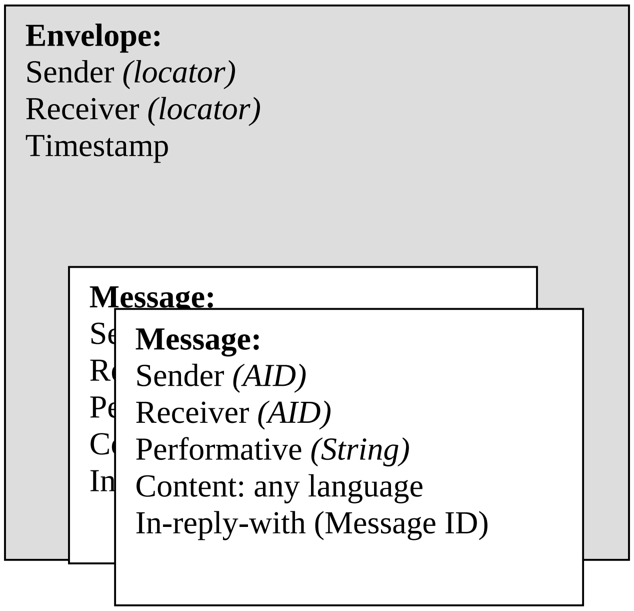


Figure 1. FIPA message structure

The Agent Management System (AMS) manages an agent life-cycle, providing a directory and transport services for all agents. A registry maintains a unique agent identifier (AID) and an agent locator. Together they specify where and how to communicate with a given agent. Agents communicate via messages structured as the key-value-tuples and are written in an Agent Communication Language.

There are many implementations of the FIPA Abstract Architecture Specification, including Java Agent Services (JAS) API (http://www.jcp.org). We begin with a brief general overview of our system - SlideTutor, and then describe our data collection framework, which builds on JAS.

SlideTutor – a medical ITS

SlideTutor (Crowley, Medvedeva, Jukic 2003) is a model-tracing ITS for teaching visual classification problem solving in surgical pathology – a medical sub-specialty in which diagnoses are rendered on microscopic slides. SlideTutor was created by adding virtual slide cases and domain ontologies to a general framework that we developed to teach classification problem solving (Crowley, Medvedeva 2003). The framework was informed by our cognitive task analysis in this domain (Crowley et al. 2003). Students examine virtual slides using multiple magnifications, point to particular areas in the slide and identify features, as well as feature qualities and feature quality values. They make hypotheses and diagnoses based on these feature sets. All knowledge (domain and pedagogic) is maintained in ontologies and retrieved during construction of the dynamic solution graph. The architecture is agent-based and builds on methods designed for the Semantic Web (Fensel et al. 1999).

An important aspect of the SlideTutor project was the development of a very generic representation for tutoring of classification problem-solving that permits all domain knowledge to be modularized into domain model, task model and case data. The domain model consists of an ontologic representation of (1) evidence as a set of features, attributes, and values and (2) the diseases to which they apply. The model is generic and can be applied to many medical diagnostic problems. The task model represents an abstraction of the goal structure for diagnostic problem-solving, for example identifying a finding or asserting a hypothesis. Data from the task model, domain model, and case are combined to create a dynamic solution graph against which student actions are tested. The underlying generic structure of task goals and subgoals and the declarative knowledge which instantiate them form the basis for collecting data about student progress through any case.

What data is collected?

The basic events collected are shown in Figure 2. InterfaceEvents record low-level human-computer interaction such as pressing a button or selecting a menu item. ClientEvents capture combinations of InterfaceEvents that represent the most atomic discrete subgoal, such as creating a hypothesis, identifying a feature, or asking for a hint. ClientEvents are answered by TutorResponses. TutorResponses indicate the response of the system to the last student action including the type of error for incorrect actions and the best-next-step at this point in the problem space (hint).

Agent-based communication protocol

Our communications system builds on JAS - a comprehensive yet lightweight FIPA standards reification. As shown in figure 1, the basic architecture encompasses three different agents: the client agent, the tutor agent, and the protocol collection agent. The client agent broadcasts low-level InterfaceEvents and higher-level ClientEvents. Interface events are sent only to the protocol agent, but client events are sent to both the protocol agent and the tutor agent. Each ClientEvent contains a set of references to the low-level InterfaceEvents that it is composed of. The tutor agent replies to any ClientEvent, sending its message to both client and protocol agents. The protocol agent captures these student-tutor interactions, transforms the messages into relational format and stores them in an Oracle 9i database. All ClientEventsare also separately stored in log files, in a format directly understood by tutor, in order to more efficiently restore system state or to provide delayed feedback. An example ClientEvent message is shown in Figure 3.

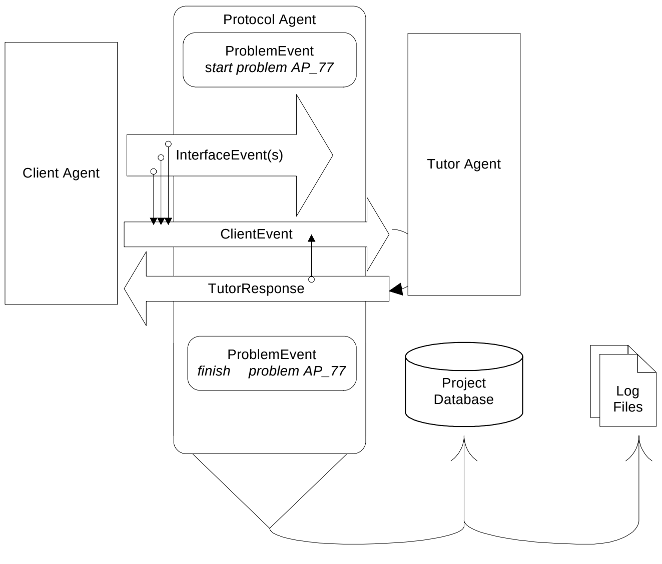


Figure 2. Agent Architecture

There are 4 required key-value pairs for any particular message within an envelope: (1) sender: AID, (2) receiver: AID, (3) content: Agent Communication Language (4) performative: defines a type of communicative act. In Figure 3, the envelope indicates the locator: the sender is a particular slide tutor client, and the receiver is the protocol agent. In this simple case, there is only one message.

The performative is a specific request from the client, in this case to create a finding of blister identified at a specific x,y,z location. The set of interface action identifiers (referencing the interface actions for (a) clicking the 'findings' button, (b) clicking in the image, and (c) selecting three times down a tree of findings) are contained in the body of the message as well.

Figure 3. Example ClientEvent

Envelope:
Sender: Client_1
Receiver: PROTOCOL
TimeStamp = 1114444377783
Message:
Sender: Concept2
Receiver: PROTOCOL
Performative: X-Create
In-reply-with: 1114444378242
Content:
Type = Finding
Label = blister
Id = Concept2
ObjectDescription = Finding.blister.Concept2
Parent = null
Input:

name = text	value = blister
name = y	value = 11808
name = x	value = 38048
name = z	value = 0.03

InterfaceEventIDS = [1114444374333, 1114444375546, 1114444376304, 1114444376798, 1114444377444]

Each interface widget is essentially its own agent, possessing a unique identifier within a particular student session. We use 5 parameters to identify each agent: (1) object Type – e.g. 'tree', 'button', 'finding', 'hypothesis', (2) Label - the name of the interface object, (3)Id – a unique object id within the current student session, (4) ObjectDescription – a combination of Type+Label+Id, e.g. 'Finding.Blister.Concept2', and (5) Parent – a list of all parent ObjectDescription for hierarchical agent structures with colon and semicolon separators within and in between the levels. Together, the ObjectDescription and parent describe hierarchical relationships between interface objects. In Figure 3, the sender of this message is actually the reified concept for the blister finding. For the interface in use when this message was generated, the object was a finding widget in the diagrammatic reasoning interface. Evolution of the interface is thus easily accommodated, as any new interface object only needs to be associated with its own set of performatives and content input.

The FIPA message structure can easily reproduce the classic Action:Selection:Input triplet of cognitive tutors (Ritter, Koedinger 1996). SlideTutor uses part of the ObjectDescription as 'Selection', Performative as 'Action' and a list of Content Input as 'Input'. The Dormin message format provides a way to communicate these Action:Selection:Input triplets across a system, but is limited because the recipient must translate the message into a format understood by the recipient. In contrast, the content of the FIPA message structure uses XML as a middle-layer, and is thus able to represent other languages (e.g. RDF (http://www.w3.org.RDF/), KIF (http://logic.stanford.edu/kif/dpans.html), SL (http://www.fipa.org/specs/fipa00008/XC00008G.html)).

Messages sent from the tutoring agent provide detail about the system response, but also the tutor state. Figure 4 demonstrates a tutor message when the ClientEvent does not match a possible next-step, according to the cognitive model. In this case an error TutorResponse is generated. The TutorResponse contains the text to be displayed in the SlideTutor student interface. But it also contains information about the type of error (error code), and indicates the best-next-step – the step that the tutor will suggest to a student request for a hint.

Figure 4.Example TutorResponse

```
Envelope Sender: TutorEngine0
Receiver: PROTOCOL
TimeStamp: 1114444379378
Message:
  Sender: TutorEngine0
  Receiver: PROTOCOL
  Performative: FAILURE
  Conversation_ID: 1114444378242
  Content:
    ErrorCode = 15
    NextStepType = Evidence
    NextStepLabel = blister
    NextStepID = 0
    NextStepParent = null
    NextStepAction = DELETE
    name = Messages
    value = "[TEXT:There is BLISTER present,
    but not where you have pointed in the
    image. See if you can find where.
    POINTERS:[PointTo:Concept2
    IsPermanent:false Method:setFlash
    Args:[true]]]"
    name= TutorAction
    value = "PointTo:Concept2
    IsPermanent:false
    Method:setBackgroundColor Args:[RED]"
```

Our categories for system responses derive directly from our ontologies for hints and errors – which maintain the semantic meaning of this error and its relationship to other errors. For example in Figure 4, the error code 15 identifies the error of indicating a correct finding in an incorrect location. The instance of this error relates to the finding 'Blister'. Once stored in relational format, the data can be analyzed in multiple ways. The query can be constructed to retrieve only errors in locating blisters, or errors of the more general type of locating findings, or all errors related to any action involving 'Blister'.

In Figure 4, the TutorResponse also contains a description of the interface object and the performative which will together describe the action that the expert model would take in this problem state. It is analogous, in some ways to the rule that fires at the end of the Match-Resolve-Act cycle. By capturing the best-next-step for each student action, we can easily and directly compare student actions over time to predictions based on the expert model. For example, we can formulate a query that returns the % of ClientEvents that match the previous best-next-step TutorResponse over time for a set of students.

Protocol agent and database schema

The relational schema we employ contains tables that closely reproduce the message structure, as well as tables and relations that allow us to maintain and retrieve data across multiple experiments, and experimental variables. Many aspects of our schema are quite similar to one described by Mostow et al (Mostow et al. 2002). Like this schema, we have tables that reflect static high-level relationships, for example a many-to-many relationship between students and experiments. Data in tables for Experiment, Experimental Condition, Case List, Tutor Case and Student do not change during a student participation in the experiment. In addition to this high-level information about experiments, students, and problems, the protocol agent also captures content of the InterfaceEvent, ClientEvent, and TutorReponse messages as well as messages created at the start and end of problems and sessions. The relational schema for this information closely reproduces the structure of the FIPA messages. The tables and relationships in our schema that describe message-based inputs are shown in Figure 5.

For each student session the protocol agent collects all reasonable interaction events, excluding window resizing and some mouse movements. The representation of these events is very generic and contains an event description with any number of possible event parameters that have

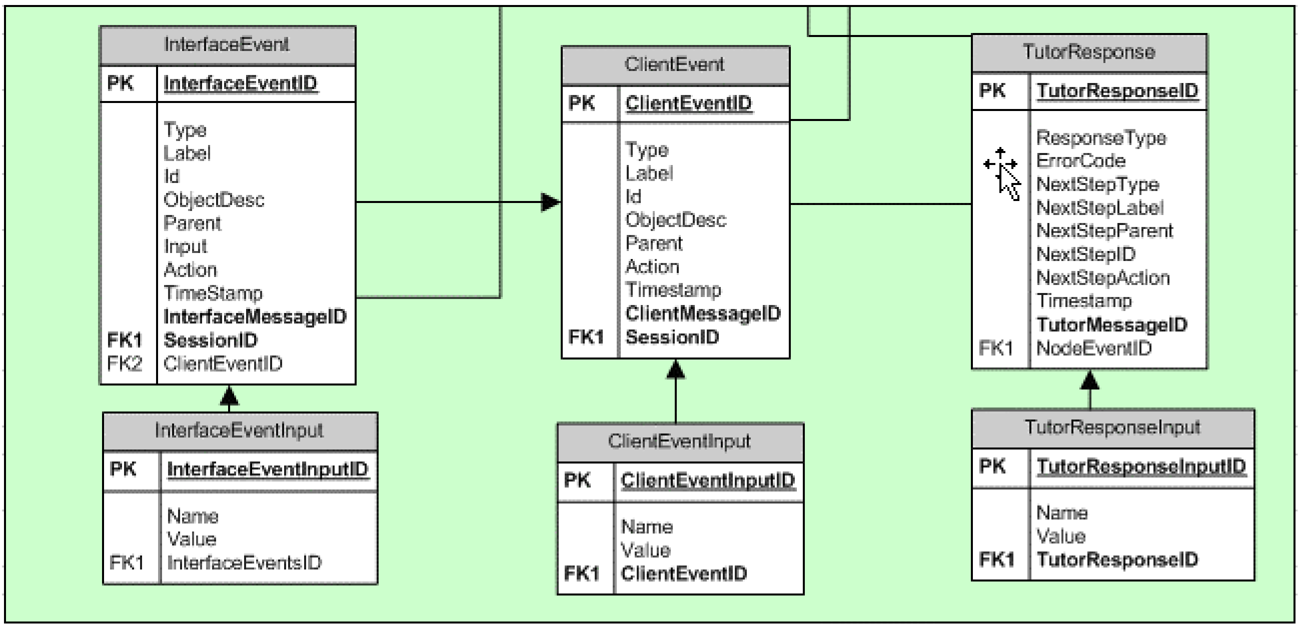


Figure 5. Database schema for low-level interaction data

been stored in corresponding Input tables. Each FIPA-based message is directly input as XML into the matching field in the appropriate table. The many-to-one relationship between InterfaceEvents and ClientEvents allows us to analyze our data from an HCI perspective as well, because we can determine for example, how many actions were performed, how much time was required to achieve a particular sub-goal such as identifying a Blister, or how many InterfaceEvents were unrelated to any ClientEvents (Saadawi et al. 2005).

Query Tool

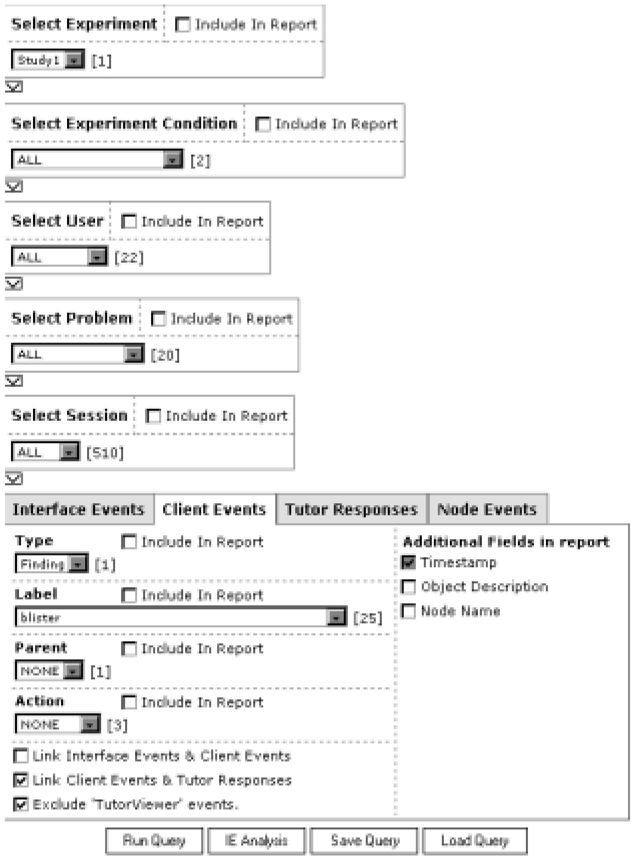
The Protocol Query Tool provides researchers with web-based querying of our database. The tool allows the user to obtain data sets specific to a wide range of constraints. The interface consists of a set of drop down lists. Each drop down list contains the set of all values existing in the database for that particular attribute. The following data elements may be specified or researchers may indicate ALL or NONE:

1. Experiment : Users may select from the set of named experiments
2. Experiment Condition : For any single experiment, users may select a particular experimental condition.
3. User: The username of the participant for the selected experiment condition
4. Problem : The case which was solved by the student during tutoring.
5. Session : The user session for the selected problem.

The researcher specifies the value constraints for the above attributes in the order shown above. As the user specifies each value constraint, the remaining attribute:value lists are updated to reflect choices for the preceding value constraints. The protocol query tool outputs HTML which can then be loaded into Excel or a statistics program for further analysis. Queries may be saved using SQL representation and then subsequently reloaded. The interface is designed to provide access to InterfaceEvents, ClientEvents, and TutorResponses for specific use-cases.

Analysis of InterfaceEvents and associated ClientTutor events is typically undertaken to analyze the usability of the system. For example, from this data we might determine (1) the total time taken to complete a particular ClientEvent as the interval from the first action which composes that ClientEvent to the last, or (2) the ratio of InterfaceEvents to ClientEvents. The tool provides a special interface to view InterfaceEvents that color codes the rows that belong to a single ClientEvent. The color coded groupings ease manual inspection by the researcher.

In contrast, analysis of ClientEvents linked to associated TutorResponses is typically undertaken to analyze student performance during tutoring, including classical metrics such as the number of hint requests, depth of hints, responses to hints, error frequency and distribution, as well as the values of these variables over time. Figure 6 shows the query tool, as a researcher requests data for all



Select Experiment ☐ Include In Report
Study1 [1]

Select Experiment Condition ☐ Include In Report
ALL [2]

Select User ☐ Include In Report
ALL [22]

Select Problem ☐ Include In Report
ALL [20]

Select Session ☐ Include In Report
ALL [510]

Type ☐ Include In Report
Finding [1]

Label ☐ Include In Report
blister [25]

Parent ☐ Include In Report
NONE [1]

Action ☐ Include In Report
NONE [3]

☐ Link Interface Events & Client Events
☑ Link Client Events & Tutor Responses
☑ Exclude 'TutorViewer' events.

Additional Fields in report
☑ Timestamp
☐ Object Description
☐ Node Name

Figure 6. Protocol Query Tool

cases in which 'Blister' is identified. It depicts the selection of ClientEvent criteria. The TutorResponse criteria may be specified by clicking on the TutorResponses tab and selecting the appropriate values from the lists.

Figure 7 shows typical output for InterfaceEvents, ClientEvents, and TutorResponses. Note that many InterfaceEvents have been recorded (7A), but only a subset of them are linked to four discrete ClientEvents (169814-169818). These ClientEvents (7B) indicate that the student erroneously asserted 'Blister' in a particular location, deleted the incorrect assertion, asked for help, and then asserted 'Blister' in the correct location. TutorResponses to these ClientEvents are also shown (7C).

Some lessons learned

Our data collection framework was deployed approximately one year ago. Since that time, we have used it to collect data on 4 smaller HCI studies and one larger experiment, for a total of 50 students. The most salient problem we encountered was that direct storage of data during usage produced records that we later wanted to remove. For example, when our systems crashed during an experiment, our research assistant had to restore the problem state for the student manually. All her actions were stored in the database under the student name. In this case, we had to delete her actions and merge the sessions to preserve the validity of the student data. Ultimately, we decided to maintain two instances of the database. The 'raw' database is used to capture all data for the project. After completion of each study we copy the data to another database, remove irrelevant and sometimes duplicate records, and then make it available for all subsequent analyses.

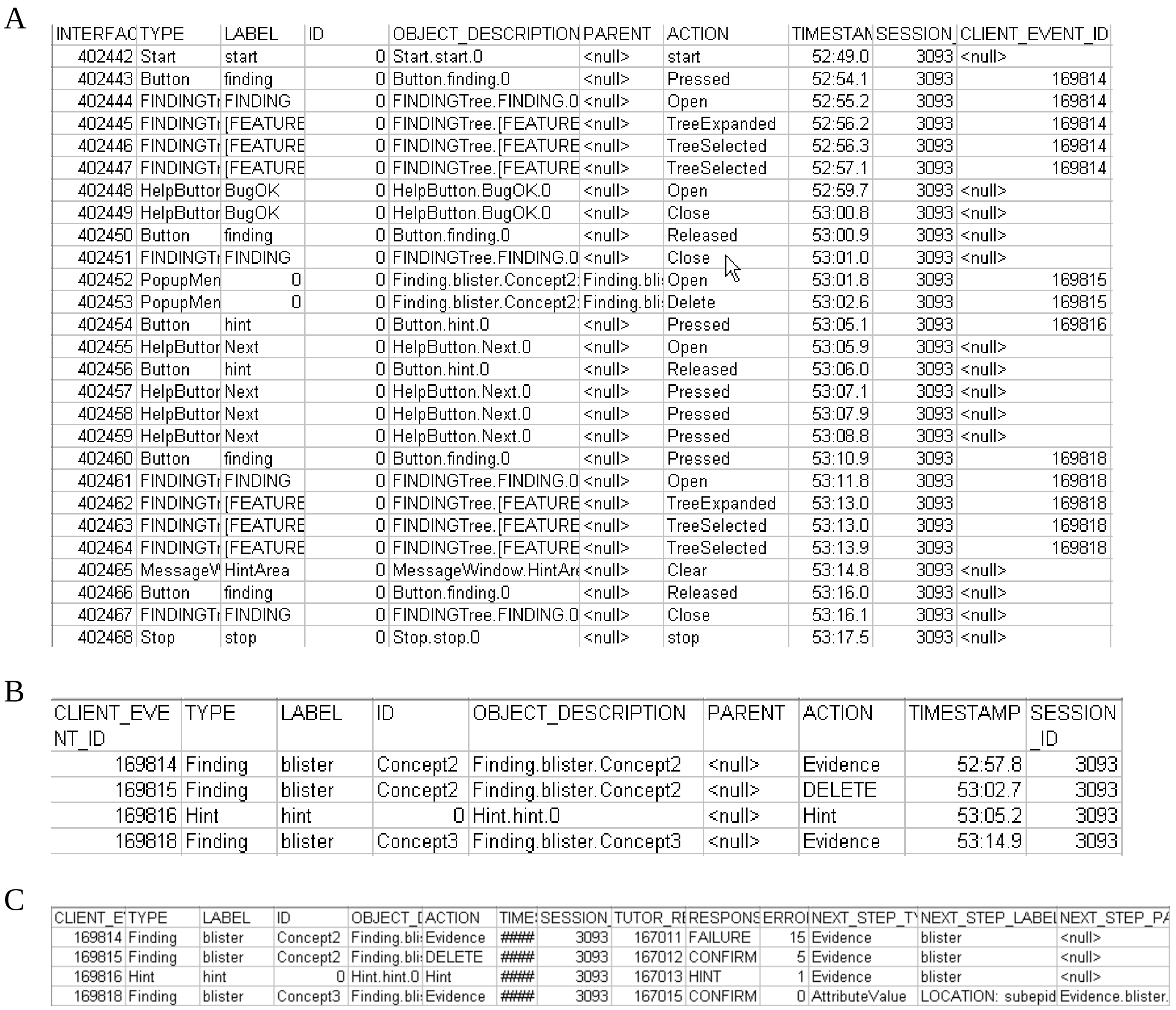
Another issue we have grappled with is that detailed real-time capture of all interaction data necessarily incurs a performance cost. In our system, every change of a virtual microscope viewer position or zoom results in a ClientEvent message that requires a reply from the Tutor. This slightly, but noticeably slows tutor response. For the detailed laboratory studies that we perform, this performance cost seems justified. However as we move our tutoring system out of the laboratory – we will need to significantly scale back the detail of the interactions that we capture in order to optimize performance.

An important limitation we discovered in our current data representation is that we are maintaining all of our assessment data in a separate database. Furthermore our assessment database does not explicitly map performance on assessments (multiple choice and diagnostic case tests) to the same skills that the tutoring system teaches. Currently, correlation of process data from tutoring sessions and performance data from assessments is a manual task. Ultimately we would like to use the system for data-mining, in order to discover novel relationships between behaviors in the tutoring system and test performance. This functionality would benefit from a unified representation of these two types of data.

Discussion and future directions

The data collection framework we describe is quite generic and might be adaptable to other model-tracing ITS. Previous authors have already cited the advantages of relational formats for maintaining study data in the design of ITS (Mostow 2002). And we concur that the single project database has helped us greatly to make sense of the enormous volume of complex data that we are collecting.

Beyond the existing implementation, we found the FIPA standard to be a very useful structure to describe ITS interaction events. The flexibility to add performatives, and the potential to reference ontologies within the FIPA message structure are particularly intriguing.



A

INTERFAC	TYPE	LABEL	ID	OBJECT_DESCRIPTION	PARENT	ACTION	TIMESTAN	SESSION_	CLIENT_EVENT_ID
402442	Start	start	0	Start.start.0	<null>	start	52:49.0	3093	<null>
402443	Button	finding	0	Button.finding.0	<null>	Pressed	52:54.1	3093	169814
402444	FINDINGTr	FINDING	0	FINDINGTree.FINDING.0	<null>	Open	52:55.2	3093	169814
402445	FINDINGTr	[FEATURE	0	FINDINGTree.[FEATURE	<null>	TreeExpanded	52:56.2	3093	169814
402446	FINDINGTr	[FEATURE	0	FINDINGTree.[FEATURE	<null>	TreeSelected	52:56.3	3093	169814
402447	FINDINGTr	[FEATURE	0	FINDINGTree.[FEATURE	<null>	TreeSelected	52:57.1	3093	169814
402448	HelpButtor	BugOK	0	HelpButton.BugOK.0	<null>	Open	52:59.7	3093	<null>
402449	HelpButtor	BugOK	0	HelpButton.BugOK.0	<null>	Close	53:00.8	3093	<null>
402450	Button	finding	0	Button.finding.0	<null>	Released	53:00.9	3093	<null>
402451	FINDINGTr	FINDING	0	FINDINGTree.FINDING.0	<null>	Close	53:01.0	3093	<null>
402452	PopupMen	0	0	Finding.blister.Concept2:	Finding.bli:	Open	53:01.8	3093	169815
402453	PopupMen	0	0	Finding.blister.Concept2:	Finding.bli:	Delete	53:02.6	3093	169815
402454	Button	hint	0	Button.hint.0	<null>	Pressed	53:05.1	3093	169816
402455	HelpButtor	Next	0	HelpButton.Next.0	<null>	Open	53:05.9	3093	<null>
402456	Button	hint	0	Button.hint.0	<null>	Released	53:06.0	3093	<null>
402457	HelpButtor	Next	0	HelpButton.Next.0	<null>	Pressed	53:07.1	3093	<null>
402458	HelpButtor	Next	0	HelpButton.Next.0	<null>	Pressed	53:07.9	3093	<null>
402459	HelpButtor	Next	0	HelpButton.Next.0	<null>	Pressed	53:08.8	3093	<null>
402460	Button	finding	0	Button.finding.0	<null>	Pressed	53:10.9	3093	169818
402461	FINDINGTr	FINDING	0	FINDINGTree.FINDING.0	<null>	Open	53:11.8	3093	169818
402462	FINDINGTr	[FEATURE	0	FINDINGTree.[FEATURE	<null>	TreeExpanded	53:13.0	3093	169818
402463	FINDINGTr	[FEATURE	0	FINDINGTree.[FEATURE	<null>	TreeSelected	53:13.0	3093	169818
402464	FINDINGTr	[FEATURE	0	FINDINGTree.[FEATURE	<null>	TreeSelected	53:13.9	3093	169818
402465	MessageW	HintArea	0	MessageWindow.HintAre	<null>	Clear	53:14.8	3093	<null>
402466	Button	finding	0	Button.finding.0	<null>	Released	53:16.0	3093	<null>
402467	FINDINGTr	FINDING	0	FINDINGTree.FINDING.0	<null>	Close	53:16.1	3093	<null>
402468	Stop	stop	0	Stop.stop.0	<null>	stop	53:17.5	3093	<null>

B

CLIENT_EVENT_ID	TYPE	LABEL	ID	OBJECT_DESCRIPTION	PARENT	ACTION	TIMESTAMP	SESSION_ID
169814	Finding	blister	Concept2	Finding.blister.Concept2	<null>	Evidence	52:57.8	3093
169815	Finding	blister	Concept2	Finding.blister.Concept2	<null>	DELETE	53:02.7	3093
169816	Hint	hint	0	Hint.hint.0	<null>	Hint	53:05.2	3093
169818	Finding	blister	Concept3	Finding.blister.Concept3	<null>	Evidence	53:14.9	3093

C

CLIENT_E	TYPE	LABEL	ID	OBJECT_[	ACTION	TIME:	SESSION_	TUTOR_RI	RESPONS	ERROI	NEXT_STEP_T'	NEXT_STEP_LABEI	NEXT_STEP_PA
169814	Finding	blister	Concept2	Finding.bli:	Evidence	####	3093	167011	FAILURE	15	Evidence	blister	<null>
169815	Finding	blister	Concept2	Finding.bli:	DELETE	####	3093	167012	CONFIRM	5	Evidence	blister	<null>
169816	Hint	hint	0	Hint.hint.0	Hint	####	3093	167013	HINT	1	Evidence	blister	<null>
169818	Finding	blister	Concept3	Finding.bli:	Evidence	####	3093	167015	CONFIRM	0	AttributeValue	LOCATION: subepid	Evidence.blister.

Figure 7. DB extract showing (A) InterfaceEvents, (B) ClientEvents , and (C) TutorResponse

What would it take to utilize this specification to create an information exchange standard for ITS? First, a set of researchers would need to agree to adopt and implement such a common message representation. Second, the group would need to develop a set of performatives that describe the common communications acts for these kinds of systems. This would, in effect, produce a controlled vocabulary for ITS communications. It might include acts such as create, delete, set-property, or show-example.

Once the basic infrastructure is in place, more complex forms of interoperability between systems could be supported using the FIPA standard. For example, error categories, hint content, or domain knowledge specified in publicly available ontologies could be referenced in the ontology attribute of the FIPA message structure. Ultimately, researchers would need to commit to standing up common services that could generalize beyond their own project to process external data. Discovery of shared services could eventually be based on an agent ontology that indicates what operations the agent performs; what inputs it expects, and what outputs it produces.

What benefits might result from this approach? First, the move to a common communications standard might produce a direct effect on data sharing, because systems would be at least syntactically and, to a smaller degree, semantically aligned. This would significantly ease some kinds of meta-analysis, because the major kinds of ITS actions would share identical performatives (for example failure, or confirm). Second, it would allow us to use each others data for other purposes such as simulation. Third, it could provide a platform for real-time system interoperability. For example, individual modules of separate ITS could interoperate around well-defined tasks such as calculating knowledge tracing probabilities or identifying particular help-seeking behaviors.

In general, discussions of standards tend to produce a mixture of fear and boredom. But one of the many lessons we have learned as ITS researchers who also work in other areas of medical informatics, is that standards act as a bootstrap to collaboration. Once individuals begin to describe their data in common ways, communication which was previously impossible becomes only difficult. Individual ITS developers appear to have an enormous amount to learn from each other. It seems worth the pain to consider how we might create our systems in ways that facilitate this kind of collaboration.

Acknowledgements

This work was supported by grants from the National Library of Medicine (R01 LM007891-01) and National Cancer Institute (R25 CA101959). The authors wish to thank Eugene Tseytlin and Katsura Fujita for their assistance with this project. Additionally, we thank Neil Heffernan for several helpful discussions during the course of developing this system.

References

1. Anderson JR, Corbett AT, Koedinger KR, Pelletier R. 1995. Cognitive Tutors: Lessons learned. *Journal of the Learning Sciences* 4(2):167-207.
2. Mitrovic A, Mayo M, Suraweera, P and Martin, B. 2001. Constraint-Based Tutors: A Success Story. In Monostori, L. and Vancza, J. (Eds). *Proceedings of the 14th International Conference on Industrial & Engineering Applications of Artificial Intelligence and Expert Systems*, Budapest, Hungary, Springer, 931-940 Budapest, Hungary: Springer.
3. Foundation for Intelligent Physical Agents: http://www.fipa.org
4. Brusilovsky, P., Ritter, S., Schwarz, E. 1997. Distributed intelligent tutoring on the Web, *Proceedings of AIED'97, the Eighth World Conference on Artificial Intelligence in Education,* 482-489. Amsterdam. IOS .
5. Ritter, S. 1997. PAT Online: A model-tracing tutor on the World-wide Web, In *Proceedings of the workshop "Intelligent Educational Systems on the World Wide Web," 8th World Conference of the AIED Society.*
6. Koedinger KR, Suthers DD, Forbus KD. 1999. Component-based construction of a science learning space: A model and feasibility demonstration. *International Journal of Artificial Intelligence in Education:* 10, 392-31.
7. Ritter, S. Koedinger, K. R. 1996. An architecture for plug-in tutor agents. *Journal of Artificial Intelligence in Education*, 7, 315-347.
8. http://ctat.pact.cs.cmu.edu/tikiindex.php?page=DorminMessages
9. Ritter S, Blessing S, Wheeler L. 2003. Authoring tools for component-based learning environments. In T. Murray, S. Blessing and S. Ainsworth (Eds.), Authoring Tools for Advanced Learning Environments, 467-489. Boston: Kluwer.
10. http://www.jcp.org
11. Crowley RS, Medvedeva O, Jukic D. 2003. SlideTutor – A model-tracing Intelligent Tutoring System for teaching microscopic diagnosis. IOS Press: Proceedings of the 11th International Conference on Artificial Intelligence in Education, 157-164. Sydney, Australia.
12. Crowley RS, Medvedeva OP. 2003. A General Architecture for Intelligent Tutoring of Diagnostic Classification Problem Solving. Proc AMIA Symp, 185-189.
13. Crowley RS, Medvedeva O. 2005. An intelligent tutoring system for visual classification problem solving. Artificial Intelligence in Medicine, (in press).
14. Crowley RS, Naus GJ, Stewart J, Friedman CP. 2003. Development of Visual Diagnostic Expertise in Pathology – An Information Processing Study. J Am Med Inform Assoc 10(1):39-51.
15. Fensel D, Benjamins V, Decker S, et al. 1999. The Component Model of UPML in a Nutshell. Proceedings of the First Working IFIP Conference on Software Architecture (WICSA1), San Antonio, Texas: Kluwer.
16. Mostow J, Beck J, Chalasani R, Cuneo A, Jia P. 2002. 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.
17. Saadawi G, Legowski E, Medvedeva O, Chavan G, Crowley RS. 2005. A method for automated detection of usability problems from client user interface events. Submitted to Proceedings of the American Medical Informatics Association Symposium 2005.
18. http://www.w3.org/RDF/
19. http://logic.stanford.edu/kif/dpans.html
20. http://www.fipa.org/specs/fipa00008/XC00008G.html